SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Games Engineering

Data-adaptive Architectures and Training Methods for Memory-efficient Volume Scene Representation Networks

Maarten Bussler

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATIK

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Games Engineering

Data-adaptive Architectures and Training Methods for Memory-efficient Volume Scene Representation Networks

Daten-adaptive Architekturen und Trainingsmethoden für speicher-effiziente Volume Scene Representation Networks

Author:	Maarten Bussler
Supervisor:	Prof. Dr. Rüdiger Westermann
Advisor:	M.Sc Kevin Höhlein
Submission Date:	15.03.2023

I confirm that this master's thesis in games engineering is my own work and I have documented all sources and material used.

Munich, 15.03.2023 Maarten Bussler

Acknowledgments

I want to acknowledge

- ... Kevin Höhlein, my advisor, for his availability, support and time he put into his helpful feedback,
- ... Luca Hohmann, Joshua Weggartner and Hendrik Rusch for proofreading,
- ... my friends and family for always keeping my mood up.

Abstract

Despite extensive research in the field of scientific data compression, modern visualization tasks encounter the obstacle of managing both memory and network bottlenecks when visualizing scientific data of high resolution at a smooth and responsive visual feedback rate. Motivated by ever increasing data sizes, techniques to efficiently compress and decompress scientific data are much in demand. Recently, compression methods that utilize neural networks have gained considerable attention in the visual computing research community. These scene representation networks (SRN), represent encoded data implicitly as an learned function and possess strong compressive capabilities by limiting the complexity of the encoding network. This thesis proposes to increase the compressiveness of learned data representations by using pruning algorithms in combination with neural networks, thereby enabling the network to learn the best tradeoff between network size and reconstruction quality during training. Furthermore, a frequency encoding of a network's feature space is proposed to enhance the pruning capabilities of dropout methods and further improve the compression quality of SRN.

Contents

1. Introduction

Voluminous scalar fields are subject to analysis in many scientific fields, such as astronomy, geoscience, neuroscience or robotics and are often generated by large scale scans or simulations. With increasing computational power, modern hardware can generate high resolution 3D data of several terabytes in size to accurately depict features of interest throughout the data domain. While this explosion of data opens up more accurate and large scale research opportunities, it also leads to significant challenges in data storage, processing, and transmission. Consequently, scientific and visual data processing tasks are faced with the problem of how to handle such complex data while providing the smooth interactivity and visual feedback needed to perform analytical tasks. To address these challenges, data compression schemes have been proposed to obtain more compact and manageable data representations to speed up data transmission and processing and enable more efficient data analysis. Popular compression algorithms include ZFP [Lin14] or TTHRESH [BLP19], which sacrifice some data precision for a better compression. However, these algorithms often rely on specific properties of the underlying data to be effective, such as a rectilinear structure or an excess of local low-frequency detail in the data. As an alternative, scene representation networks (SRN) have gained prominence in the field of visual computing in recent years. These fully-connected neural networks implicitly represent encoded fields by learning a function that maps spatial positions from the encoded domain to a decoded scalar value. By limiting the complexity of the network (e.g. the amount of parameters in the network) to be smaller than the resolution of the original data, these networks themselves act as compressed versions. Neural networks make few assumptions about the characteristics of the underlying data. Instead, during training the network learns the most important features to approximate a given field and is able to reconstruct the encoded data at arbitrary resolution. Scene representation networks are also able to provide random access to the data, allowing for efficient processing and visualization of large data sets. Moreover, these networks are able to exploit non-local coherence of the data, enabling to exploit complex relationships and structures that may be missed by conventional compression methods.

Motivated by the success of SRN in visual computing tasks, this thesis investigates possibilities of further increasing the compressiveness of learned data representations by using pruning and dropout algorithms in combination with neural networks. While

dropout was originally introduced to mitigate overfitting in neural networks [Hin+12], recent research suggest that it can also be utilized as a means of network compression. By utilizing pruning and dropout algorithms, a network can learn which internal nodes and features are most critical to reconstruction quality. The network can then remove the uncritical nodes from the network architecture, thereby learning the best tradeoff between network size and reconstruction quality during training.

Specifically, this thesis analyzes the effects of pruning methods such as Sparsify [Lec+18] and variational dropout [KSW15] on modern SRN such as Neurcomp [Lu+21] and fv-SRN [WHW22]. Moreover, this work proposes a frequency encoding of a network's feature space to enhance the pruning capabilities of dropout methods and further enhance the compression quality of SRN.

This thesis is organized into seven chapters. In Chapter 2, preliminary knowledge about neural networks and data compression is introduced. Chapter 3 presents a comprehensive review of the existing literature on SRNs and network compression. Chapter 4 reviews the implementation of Neurcomp and fv-SRN, as well as the utilized dropout algorithms. In Chapter 5, experiments on the improvements of SRNs and the internal workings of the dropout methods are depicted. Finally, chapter 6 discusses and reviews the experiments and chapter 7 presents a conclusion summarizing the findings.

2. Preliminaries

In this chapter preliminary knowledge about scientific data, neural networks and scientific data compression is explored.

Specifically, the network architectures of Neurcomp [Lu+21] and fv-SRN [WHW22] for implicit data compression are introduced.

2.1. Scientific Volumetic Data

To understand the process of volume compression, it is first important to realize the general structure of volumetric data sets. Many visual effects (e.g., clouds, gases, or liquids) are volumetric in nature with complicated or changing topologies and are difficult to model with geometric primitives. As a solution, a specific form of representation, the volume data set, was introduced. They are 3D entities that do not consist of either tangible surfaces or edges and characterize the entire space that encloses an object.

Volumetric data is typically understood as a set S of samples (x, y, z, v) of a continuous scalar field over the examined space, where a value v represents a property of the data (e.g., color, density, heat, or pressure) at a 3D position (x, y, z) [SF+00]. In most cases, samples are taken at regular intervals along three orthogonal axes. A 3D grid or tensor is usually used to store S, where the position of the element indicates the position of the sample on the grid. Since the grid form of S is only defined at discrete grid locations, various interpolation techniques such as trilinear or bicubic interpolation are used to generate values at arbitrary continuous positions.

Volume data is obtained by numerous sampling, simulation and modelling techniques and are thus of great interest for scientific research. In the medical field, MRI and CT-Scans acquire data as 2D slices, which are later reconstructed in a volumetric fashion for the purpose of better visualization. Many computational fields, like fluid dynamics and computer vision tasks, such as semantic scene segmentation and classification, also depend on volume data, as the results of simulations are often visualized as 3D volumes for later analysis and interpretation [HJ11].

2.2. Compression Techniques for Scientific Data

Scientific volumetric data is often the result of simulations or scans that produce high dimensional scalar fields represented as floating points. Interactivity and fluent visual feedback between the data set and the data consumer are critical for visualizing data in real time and crucial for performing efficient analysis tasks. With recent technological advances, scanners and computer simulations are becoming more powerful and produce very explicit and in-depth volume data. Especially for scientific applications that work predominantly with large arrays of floating-point numbers, the size of such large data sets and the required loading and processing times pose a number of challenges in terms of visualization, bandwidth constraints, data storage, accessibility and interactivity. Data compression attempts to address these I/O-related difficulties by reducing the size of records in memory and reducing transfer times to and from disk. In general, data compression techniques can be assigned to one of two groups [Li+18]:

- **Lossless** compression is achieved by using patterns in the data. No information is intentionally discarded, and apart from floating-point rounding errors, the reconstructed data after compression will be the same as the original data.
- **Lossy** compression reduces the data by truncating insignificant values in the data set. Therefore, the reconstructed data does not match the original data perfectly, but much higher compression rates can be achieved than with lossless methods. While these techniques generally attempt to minimize the introduced error, the extent of information loss varies greatly from application to application, and often the exact reduction parameters need to be modulated on a case-by-case basis.

For dealing with scientific data, lossy compression is often the prescribed strategy. This policy is based on the observation that many applications and simulations, as well as the floating-point format itself, allow for some error and can get by with far less accuracy than originally intended. Furthermore the voluminous nature and huge storage footprint of this data often requires the additional compression of lossy methods to be effective. Unlike images and videos, scientific data generally requires the construction of visual representations and an explicit rendering of those representations for visualization. Typical representations include meshes, point clouds or volumetric voxel grids, which can be obtained from scalar fields by isosurface extraction [LC87] or by assigning colors and opacity values via transfer functions of direct volume rendering.

These representations are often compressed by exploiting spatial and temporal coherence in the data and by applying sparsifying transforms. Early work in this area focused on applying a series of Fourier decompositions and later wavelet transforms (JPEG2000 [TM02]) to the data, separating low and high frequency details and allowing

compression by discarding high frequencies while preserving low, basic frequencies. TTHRESH [BLP19] extends this idea and applies a singular value decomposition on tensors to greedily compress data subparts of progressively less importance. Other important compression algorithms include SZ [DC16] or ISABELA [Lak+13], which linearize multidimensional data and then replace data entries with a best-fit curve-fitting model or monotone B-splines that accurately predict successive data points. Another widely used compression algorithm is ZFP [Lin14], which compresses high-dimensional floating-point arrays by dividing the large array data into smaller cubic blocks. Each block is then compressed individually by applying a transform to the grid values, ordering the transformation coefficients by size, and truncating insignificant bits of the smaller coefficients.

While effective, the success of these compression methods depends on the structure of the underlying data, e.g. if the data is composed of largely low-frequency fourier bases or admits low-rank decomposition for other transform based approaches. These compression algorithms also often impose some constraints on random access ability of the data, thereby further limiting their use for interactive applications.

While research on traditional data representation formats such as 3D meshes, point clouds, and volumetric data has led to great advances in areas such as 3D scene understanding and scene interaction, these data formats are limited by their achievable spatial resolution in regards to their memory footprint and the need for explicit 3D supervision for generation. This thesis aims to leverage the increasing interest in applying machine learning principles to data visualization tasks and examine methods for developing learning-based compression algorithms. Specifically, the work in this thesis is interested in continuous Neural Scene Representation Networks (SRN) that model both voluminous 3D scene geometry and appearance implicitly as a continuous, differentiable function. By mapping a 3D world coordinate to a feature-based representation of the scene properties at that coordinate, SRN are able to randomly access their encoded data, while operating at arbitrary spatial resolution in a memory-efficient manner. Compressing such implicit scene representations provides the opportunity to create a compression scheme that takes advantage of both the global and local coherence of the data and makes little assumptions about the properties or structure of the underlying data. Although these learning based compression methods can achieve remarkable results on their own, they are often coupled with traditional, non-trainable compression algorithms to enhance the encoding of their network parameters.

The following subsections provide a general overview of the non-learning based compression algorithms discussed in this thesis. Each technique has different tradeoffs in terms of compression ratio, Processing time, and susceptibility to errors.

2.2.1. TTHRESH

TTHRESH [BLP19] is a lossy thresholding algorithm that combines low approximation errors with smooth and dynamic target compression ratios. The compression algorithm leverages the higher-order singular value decomposition (*HOSVD* [BP16]), a tensor decomposition that generalizes the 2D matrix SVD to higher dimensional tensors, to approximate a given input data set by a set of orthogonal factor matrices and a core tensor. The algorithm then applies adaptive *bit-plane coding*, *run-length coding* and *arithmetic coding* to compress the underlying data by thresholding *HOSVD* transform coefficients and bit planes of less importance. TTHRESH achieves competetive quality-compression results and outperforms other compressors, such as ZFP [Lin14] and SZ [DC16] in terms of reconstruction quality, especially at higher target compression ratios. However, the algorithm is slowed down by its inflexible approach to compressing the SVD matrices and the core as a whole, which increases the cost of random access decompression and results in comparatively slow reconstruction time.

In this work TTHRESH is used as a baseline for modern compression algorithms without neural networks.

2.2.2. Quantization

Quantization is a lossy compression technique that limits the complexity of the input data by reducing the precision of the data values [Li+18]. Quantization is rarely used on its own, but forms the basis for numerous complex compression methods and is usually combined with other compression algorithms. In this thesis quantization is used on SRN to limit the precision of the network parameters and to reduce the required storage size, thereby increasing the compression rate of the network at the cost of reconstruction accuracy.

Compression is achieved by mapping from floating point values to a finite set of fixed output values (*bins*), which serve as approximations of the input data (Figure 2.1). For example, the rounding of floating point values to integer values could be considered a simple form of a quantization process. The device or function that performs the specific mapping of the data is called a *quantizer*, while the difference between the original value and the mapped value (e.g. the rounding error in case of rounding floating points to integers) is referred to as the quantization error. Considering the many-to-few mapping of the input data, quantization is a nonlinear and irreversible process. As a result, it is impossible to recover the exact original value of the input data, since numerous input values are assigned the same output values.

Input data sets often differ greatly in their value distributions. For this purpose, and to improve the approximation ability of the bins, the mapping schema can be adjusted

to best capture the underlying data and to reflect the uniform or non-uniform value distributions.

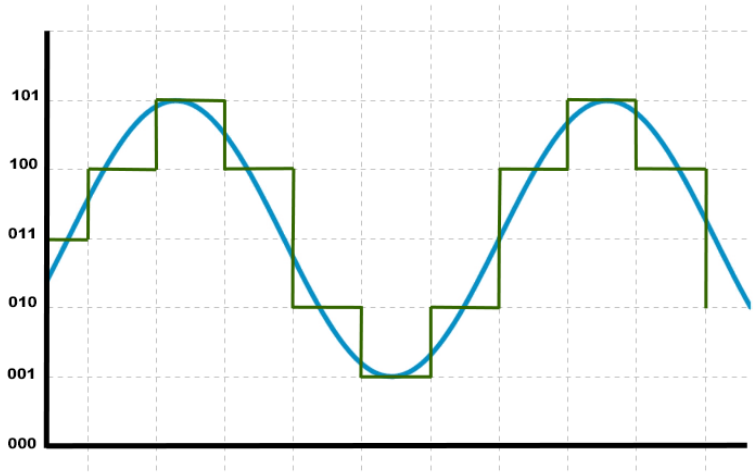


Figure 2.1.: Illustrative example of the process of quantization. A sine wave (blue) is approximated and and encoded with quantization (green). Each output value is encoded with 3 bits. Adapted from [Bha18].

2.2.3. Wavelet Transform

The wavelet transform is a technique for analyzing image, video and audio signals in the time and frequency domain. This is done by decomposing a signal into a set of components of varying frequencies and durations that perform well in capturing local features of the base signal [dCM00]. This decomposition of the base signal is advantageous for multiple compression algorithms. From a high level viewpoint, wavelet compression treats input signals as set of *wavelets* and uses *wavelet transforms* to gather information about the input file. In an abstract way, wavelets can be viewed as the changes in the input data, measured as the deviation of a single data value from zero. By recording the distance of deviation from each data point as a coefficient, a set of wavelets can then represent the original data. The process of measuring and recording these coefficients of data points is called *wavelet transform*.

Different coefficients represent distinct global or local information, and often only a handful of coefficients contain significant information of the original signal. Traditional compression algorithms leverage the wavelet transform for compression by quantiz-

ing or truncating the coefficients to varying degrees of accuracy, depending on how important the coefficient is to the reconstruction accuracy of the base signal. In this work this data sparsification approach is utilized in combination with learning-based compression schemes: By encoding the feature space of a neural network with the wavelet decomposition, the network is able to identify the most important components and can discard the rest. This results in a more memory-efficient representation of the network's feature space.

In a more mathematical form, wavelets are functions that can be used to split an input signal into different frequency bands (*sub-bands*). Just like the Fourier-Transform [BB86] represents a signal as the sum of sine and cosine signals, the wavelets can be linearly combined to synthesize a signal. Unlike the Fourier-Transform however, wavelets at different scales all derive from the same *basis wavelet* through translation and dilation [ZB21]. Given a basis wavelet $\Psi(t)$, smaller wavelets can be derived through:

$$\Psi_{a,b}(t) = \frac{1}{\sqrt{a}} \cdot \Psi(\frac{t-b}{a}) \tag{2.1}$$

where the coefficient a amounts to dilating, and b to translating the basis wavelet. The wavelet transform $X(a,b)$ can then be described as the product of the smaller wavelets and the base signal $x(t)$:

$$X(a,b) = \int_{\infty}^{-\infty} x(t) \cdot \Psi_{a,b}(t) dt \tag{2.2}$$

This approach offers the possibility to analyze signals with a much more flexible tiling of time and spatial frequency (See Figure 2.2) and provides a way to order different frequency components of the base signal in a hierarchical fashion. By combining short high frequency base function with longer lower frequency ones, wavelet transform provides a set of basis functions that can accurately and compactly represent local features, as well as the large-scale characteristics of the base signal.

The *discrete wavelet transform (DWT)* is a discrete version of the continuous wavelet transform formulation in Equation 2.2. The DWT divides a signal into a set of sub-bands by convolving the original input data with a discrete low-pass filter h and high-pass filter g of fixed length at intervals of 2. The sub-band resulting from the low-pass filter contains a basic, downsampled version of the original data, while the high-pass filter localizes the high-frequency detail of the signal. The low-pass detail can then be iteratively transformed again to form a higher level wavelet transform and to increase the frequency resolution of the wavelet representation [Nec04]. The construction of h and g depend on the underlying basis wavelet, where common a choice is the family of *Daubechies* [VBU07] wavelets, e.g. the *Haar* [SF03] wavelet.

JPEG2000 [TM02] is a concrete example of a non-trainable compression algorithm that uses wavelet transforms. The algorithm aims to improve the JPEG-compression standard with a compression method that offers a better compression rate and fewer artifacts. The algorithm first employs a set of 2D wavelet filters to transform pixel information of the input image into wavelet coefficients. The coefficients are then grouped into sub-bands and arranged hierarchically, with the low-frequency, basic information encoded in the lowest level and the higher-frequency details encoded in the higher levels. Code blocks (bit streams of data) are then generated from each sub-band and ordered according to their importance to the reconstruction quality. Compression is then achieved by quantization and truncation of the bit streams in the code blocks.

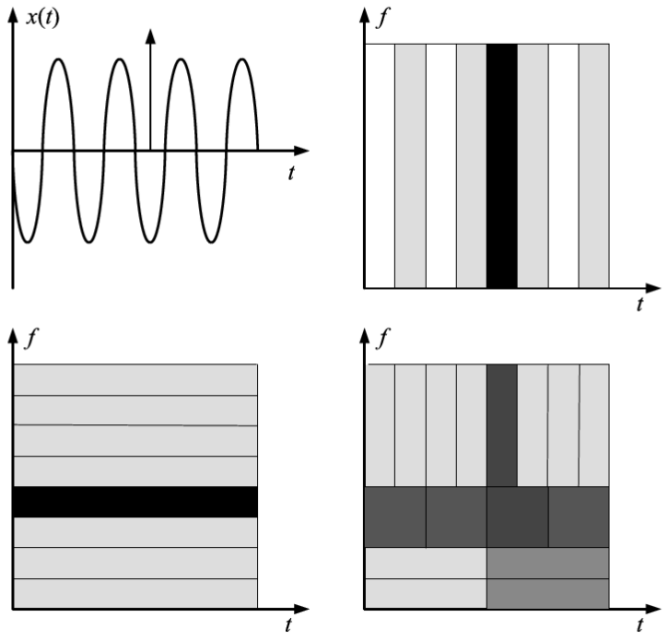


Figure 2.2.: A combination of a sine signal and an impulse function (top left) is captured in the time domain (top right), frequency domain (bottom left) and wavelets as a combination of time and frequency (bottom right). The time domain can accurately depict the impulse in time, but has troubles representing the frequency of the sinus signal. The frequency domain can localize the sinus signal, but has uncertainties in time. In contrast the wavelet transform tiles the frequency-time plane more adaptively (higher time resolution at higher frequencies) and can thus represent both short duration high frequency components and longer low frequency components of the base signal. Adapted from [ZB21].

2.3. Neural Networks

Neural networks are algorithms designed to solve problems that are difficult to define by formal mathematical rules and are therefore difficult to program directly. An example would be the tasks of speech or face recognition. In machine learning, these problems are addressed by enabling the algorithm to learn from experience and build complex concepts and functions by combining simpler functions. As such, a simple neural network consists of a set of interconnected *neurons* (*nodes*) grouped in an *input* and *output layer*. Each node encapsulates a simple linear function that is augmented with a nonlinear *activation function* to introduce nonlinear behaviour into the network. According to the universal approximation theorem [KA01], a neural network can approximate any function through its learned parameters.

Deep neural networks (DNN) are a subclass of neural networks that realize more complex functions by nesting multiple layers of neurons between the input and output layer (*hidden layers*) (See Figure 2.3). To learn a particular task, a DNN trains its network parameters by minimizing the error between the observed data (predicted by the network) and the expected data (ground truth) [CK19]. For this purpose, the total loss between the predicted and the expected data is propagated from back to front in the network and the loss gradient of each weight is calculated (called *Back-Propagation*). Then, each weight is updated according to the calculated gradient.

In this work a deep neural network is formally defined as a tuple $f = (L, T, \Phi)$, where $L = \{L_k \mid k \in \{1 \dots K\}\}$ is a set of layers consisting of s_k nodes. $T \subseteq L \times L$ are the connections between layers (*weights*) and $\Phi = \{\phi_k \mid k \in \{2 \dots K\}\}$ is a set of functions for each non-input layer. Except for inputs, every node is connected to nodes in the preceding layer, such that

$$y_{k,l} = \phi_k(b_{k,l} + \sum_{1 \leq h \leq s_{k-1}} w_{k-1,h,l} \cdot x_{k-1,h}) \tag{2.3}$$

$y_{k,l}$ is the output of the l-th node of layer k. $w_{k-1,h,l}$ represents the *weight* of the connection between the h-th node of layer $k-1$ and the l-th node of layer k, $b_{k,l}$ being the *bias* for node $n_{k,l}$, ϕ_k the nonlinear *activation function* of layer k, and $x_{k-1,h}$ the output of the previous node [Sun+18].

Neural networks are flexible and powerful tools that achieve state-of-the-art results in a wide variety of visualization tasks, such as object detection, recognition, or generation. Recent advancements in machine learning have also led to an increasing interest in representing arbitrary data through neural networks for rendering and replication along with compression [Xie+22]. One example are Scene Representation Networks (SRN) [SZW19], which represent data as a continuous function. In case of a 3D scene, this function can be trained from samples of the scene or a set of 2D input images

and poses, and encodes both geometry as well as appearance of the original scene. By limiting the number of parameters used in the network to be less than the original resolution of the scene, these approaches can obtain compressed volume representations optimized for the best approximation of the scene to its samples.

The goal of this thesis is to analyze the compressive quality and ways of optimization for two selected (SRN) architectures: Neurcomp [Lu+21] by Lu et al. and fv-SRN [WHW22] by Weiss et al. The next two subsections will examine these two network architectures more thoroughly.

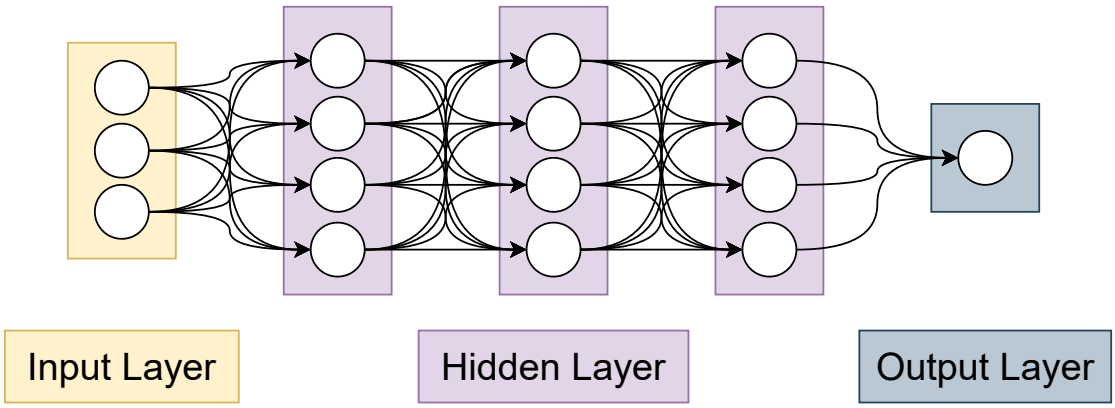


Figure 2.3.: Example of a fully connected deep neural network. The network consist of an input layer, three hidden layers and an output layer. Each layer consists of neurons that hold weighted connections to the neurons of the subsequent layer. In this example, the input layer takes three input variables and feeds them into the network. The four neurons of each hidden layers then extract and pass along features from the input of the preceding layer until the output layer interprets the extracted high level features to a scalar value.

2.3.1. Neurcomp

Neurcomp [Lu+21] is a lossy compression method to represent voluminous scalar fields with implicit neural representations. This is done by representing a scalar field with a neural network as a learned function, where the network takes samples of the field as input and maps them to scalar output values. Decoding the original scalar field from the network is then a simple process of sampling the network using ray tracing and visualizing the results using direct volume rendering. Assuming the network weights and volume are of the same precision, significant compression can be achieved by limiting the number of network parameters to less than the volume resolution. The

level of compression thus follows from the permitted amount of parameters. This compression approach makes few assumptions about the characteristics of the input data and has been shown to outperform conventional state-of-the-art compression algorithms through its consistent results in terms of compression results and approximation quality. In this work Neurcomp is used as a baseline algorithm to explore the possibilities of increasing the efficiency of compression techniques based on fully connected neural networks.

SIREN [Sit+20] has recently been shown to have good properties for representing 3D data. The work by Sitzmann and Marte proposes to leverage periodic activation functions together with fully connected layers, in order to capture a signal's higher order derivatives. Compared to traditional ReLU-based networks, this method allows for much faster and more detailed data fitting, without needed preprocessing of the data. A layer in the SIREN network is defined as

$$\Phi_i(x_i) = \sin(\omega_i W_i x_i + b_i) \tag{2.4}$$

where Φ_i is the i-th layer that consists of the weights W_i and bias b_i. The weights and biases of each layer are initialized according to an uniform initialization scheme $W_i \sim \mathcal{U}(-\sqrt{6/n}, \sqrt{6/n})$, with n describing the size of input features of the layer. This initialization makes sure that the initial prediction of the network does not depend on the number of network layers and that the input for each sinusoidal activation is normal distributed in $[-1, 1]$. ω_i serves as a scale hyperparameter that increases the spatial frequency of the first layer to better match the frequency spectrum of the underlying signal.

Figure 2.4 depicts a general overview over the Neurcomp network structure. Neurcomp builds on this approach and defines it's architecture by a sequence of fully-connected SIREN layers that use sinusoidal activation functions. The network layers are organized in residual blocks [He+16] to secure the robustness and stability of the network even for deeper models with many layers. The invention of residual blocks is based on the observation that a neural network with many layers sometimes performs worse on a prediction task than a comparably shallow and simple network. As stated before, deep neural networks excel at extracting complex features out of the input data and are able to learn complicated functions. However, in feeding the input deeper and deeper into the network layers, the network discards the input of each layer and thus looses access to the low-level basic information of the input. This behaviour can lead to an accuracy drop of the network predictions, since both low- and high level information are needed to accurately represent the original data. A residual block aims to solve this problem by routing incoming input to the next layer and directly to a deeper layer two or three layers away. By concatenating new extracted features and lower-level features from higher up in the network as input for a network layer, the

low-level features are preserved for a longer time inside the network, and stability and prediction accuracy of the network increases. In full, a residual layer in the Neurcomp network is thus defined as:

$$\Phi_i(x_{i-1}) = 0.5 \cdot (x_{i-1} + \sin(W_i^2 \cdot \sin(W_i^1 \cdot x_{i-1}))) \tag{2.5}$$

where x_{i-1} is the output of the previous residual block $i-1$, Φ_i is the current block and W_i^2 as well as W_i^1 are a pair of learnable weight matrices associated with block i.

During training, a data set X is probed at random positions p_i and the resulting spatial positions are normalized and fed into the network. The network is optimized with backpropagation, where the training loss consists of a volume loss that compares the predicted result of the network parameterized by $\Theta \in \mathcal{R}^m$ against the ground truth with a mean squared error loss:

$$\min_{\Theta} \sum_i (f_\Theta(p_i) - X[i_1, i_2, \ldots, i_d])^2 \tag{2.6}$$

Additionally, the use of sinusoidal activation functions in the network layers makes it possible to optimize for higher order derivatives of the scalar input field as well. This enables the network to also optimize for and preserve the gradient information of the original input data, in addition to the provided scalar field values. Incorporating gradient information into the network optimization process increases the reconstruction accuracy of Neurcomp, as the acquisition of higher order information is beneficial, for example, to avoid artifacts on isosurface contours. To this end, the training loss is enhanced with an optional loss term that encapsulates the difference of the predicted network gradient to the gradient field of the original data. The final training optimization thus amounts to the following equation:

$$\begin{aligned} \min_{\Theta} \sum_i (f_\Theta(p_i) - X[i_1, i_2, \ldots, i_d])^2 + \\ \lambda_g \cdot ||(\nabla f_\Theta(p_i) - X'[i_1, i_2, \ldots, i_d])||_2^2 \end{aligned} \tag{2.7}$$

where λ_g is the weighting of gradient loss to the volume loss. Optimization is then performed with the ADAM algorithm [KB14]. The user can also specify a learning rate strategy that adjusts the learning rate of the optimization during training.

After training, the compression ratio achieved by Neurcomp depends solely on the choice of network complexity as the number of parameters in the network m and can be described as $\frac{C}{m}$. The authors of Neurcomp note that the distribution of weights in each network layer roughly resemble a normal distribution. They recognize that further compression can be achieved by quantizing the network weights (see subsection 2.2.2). Similar to the deep compression approach of Han et al. [HMD15], quantization is performed by encoding the weight values of each layer with k-means clustering.

For inference and decoding of the compressed data, the quantized network parameters are first restored. The space of the original data is then probed and the spatial positions are fed into network, which predicts and reconstructs the encoded data. The reconstructed scalar values can then either be interpreted as-is, transferred into another data format, or used in combination with direct volume rendering techniques (like proposed in NERF, see subsection 2.3.2) to render a volumetric representation.

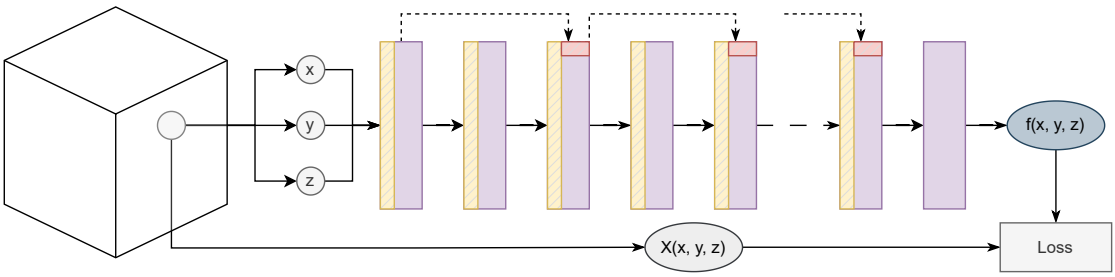


Figure 2.4.: Overview of the Neurcomp compression method. Encoding is done by tasking the network f to predict scalar values $f(x, y, z)$ with given ground truth data $X(x, y, z)$ at coordinates (x, y, z). The network is composed of a set of fully connected layers (violet boxes) with periodic activation functions (yellow boxes) and a linear layer without activation at the end. The dashed arrows and red boxes represent residual connections. Adapted from [Lu+21].

2.3.2. fv-SRN

Fast Volumetric Scene Representation Networks (fv-SRN) by Weiss et al. [WHW22] represent another compression approach for 3D scalar fields that achieves high reconstruction quality by training neural networks to implicitly represent the original data. For many conventional SRN-approaches, such as Neurcomp [Lu+21], the size of the encoding network heavily impacts the training and test time. Their usability in real-time applications is thus held back by their need for many computationally heavy and slow forward passes and long rendering time.

In contrast, fv-SRN extends research on implicit neural representations by proposing a compressive network architecture that focuses on real-time applicability. To this end, the authors of fv-SRN divide the traditional monolithic network architecture into a latent feature space and a relatively small fully connected network that utilizes the feature space for a higher reconstruction accuracy of the original data. The feature space is trained simultaneously with the rest of the network and consists of a grid that

holds feature vectors at its vertex positions. These feature vectors represent the majority of network parameters in the architecture and hold high-level semantic information about the encoded data that can be interpreted by the network. Since the feature grid contains most of the implicit expressive power of the network architecture and the fully connected network itself is quite small, Weiss et al. are able to load the network parameters into the shared memory space of a GPU. They can then use GPU tensors to integrate data reconstruction into efficient on-chip ray tracing, resulting in significantly faster decoding times. In this thesis fv-SRN is used as a baseline algorithm to investigate opportunities to use dropout algorithms in combination with smaller networks that store most of their parameters in a separated data structure, like the feature grid.

Figure 2.5 portrays the internal network architecture of fv-SRN. At training and inference time, the neural network itself is provided with a spatial input coordinate p and tasked with predicting the original scalar value at p. The network consists of a set of fully connected layers, with the i-th layer defined as:

$$\Phi_i(x_i) = \phi(W_i \cdot x_i + b_i) \tag{2.8}$$

where x_i is the output of the previous layer and input to the current layer, W_i and b_i are the weights and biases of the current layer and represent the trainable parameters of the network. ϕ represents a non-linear periodic activation function, similar to the sinusoid SIREN activations in Neurcomp (see subsection 2.3.1). The authors of fv-SRN reason that the *SnakeAlt* function slightly improves the prediction quality of the network compared to the standard ReLU activations:

$$SnakeAlt(x) = 0.5 \cdot x + \sin^2(x) \tag{2.9}$$

Additionally, fv-SRN employs a high frequency input embedding, to further enhance the reconstruction quality of the network. The idea of a supplementary input encoding stems from work of Basri et al. [Ron+19] as well as Rahaman et al. [Rah+19] showing that neural networks are biased to learn lower frequency functions easier than higher frequency functions. This leads the network to first fit to the low frequency components of the target function, and then to the higher frequency parts. If data is missing and needs to be predicted, the Network as such interpolates with a low-frequency function instead of a more straightforward curve that would introduce higher frequency components. The network itself acts as a smoothing filter on the original data, that needs to be accounted for. Consequently, most scene representation networks without any form of frequency embedding result in renderings that only poorly capture high frequency variation in color and geometry. To counteract this issue, the authors of NeRF [Mil+20] propose to leverage high frequency functions to map the original input to a higher dimensional space before passing it to the network. They point out that

this mapping increases computation time, but also enables the network to better fit to higher frequency components of the data and significantly increases rendering quality. In the case of fv-SRN, Weiss et al. decide to enrich the input coordinate p with a Fourier Feature encoding γ that applies sinusoids of varying frequency on p:

$$\gamma(p) = [\sin(2^0\pi p), \cos(2^0\pi p), \cdot, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)]$$

where L is a hyperparameter describing the number of encoding frequencies.

Lastly, fv-SRN employs a three dimensional discrete latent feature grid that holds feature vectors at it's vertex positions. This approach originates from the observation that a Scene Representation Network small enough to fit in the shared memory of a GPU to allow interactive rendering does not have the expressive complexity to accurately represent the encoded data. To solve this problem, the feature grid and its feature vectors are trained simultaneously with the small network and used to store semantic information about the encoded data that can be utilized by the network to make accurate predictions. When training and evaluating the network, the feature grid is thus interpolated at position p and the resulting feature vector is passed to the predictor network as additional input, along p and $\gamma(p)$.

The latent feature grid and network layers are trained with backpropagation, where a volume loss similar to the mean squared error loss applied in Neurcomp (see Equation 2.6) is used. Inference and reconstruction of the original data is also similar to Neurcomp: the latent feature grid and parameters of the fully connected layers are loaded from memory and used to predict the scalar values of the encoded data. These can then be combined with direct volume rendering techniques to reconstruct the original data.

2.4. Pruning of Neural Networks

Neural networks achieve great success in many visualization tasks and are a topic of great interest in modern artificial intelligence research [LSV19]. To generate neural networks with high accuracy, these works often rely on deep architectures with many layers and millions of parameters. The high number of parameters make DNN memory intensive, and access to GPUs with high computational power and long training times are essential for the success of these types of networks. This hinders the deployment of deep learning systems in portable devices with limited memory and computational power, as well as in real-time applications with stringent latency requirements. As a result, much research has been conducted to rethink established data compression algorithms for application to neural networks in order to achieve model compression and acceleration without significant degradation of model performance.

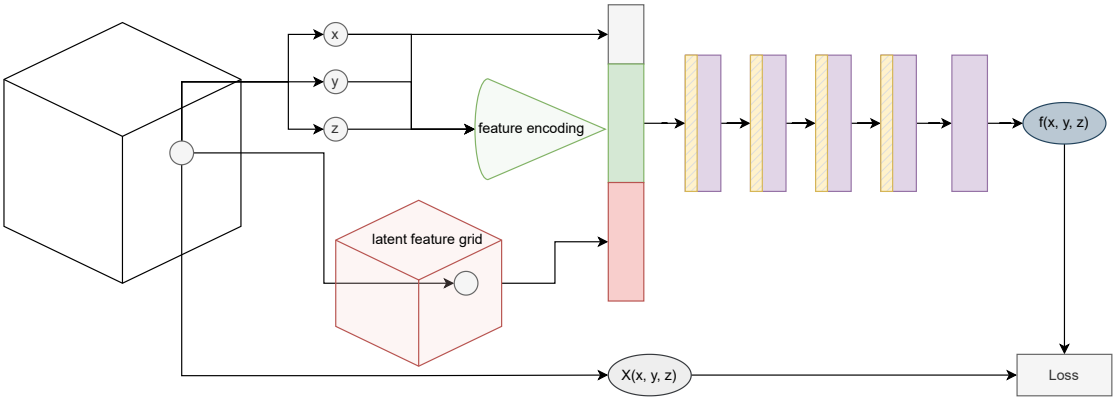


Figure 2.5.: Overview of the fv-SRN architecture. Given data X and a spatial position (x, y, z), the network f predicts a scalar value $f(x, y, z)$ that is compared against the ground truth $X(x, y, z)$ for training. The network is composed of a small set of fully connected layers (violet boxes) with *SnakeAlt* activation functions (yellow boxes) and a linear layer without activation at the end. The input for the network layers consists of the spatial position (x, y, z), a feature encoding of (x, y, z), and a feature vector from a latent feature grid interpolated at (x, y, z).

Recently, parameter pruning and dropout based methods have shown great success in the area of neural network compression [Che+17]. These algorithms examine the redundancy in the model parameters and attempt to remove the redundant and non-critical parameters. Although different pruning strategies exist, most modern pruning algorithms derive from the same high-level algorithm [Bla+20]: First the network is trained to convergence. Afterwards, scores are issued to the network parameters, and the network is pruned based on these scores. As pruning reduces the accuracy of the network, it is often trained further (known as *fine-tuning*) to recover.

This thesis investigates the potential of pruning algorithms as a primary tool for reducing network complexity and optimizing the compressive capabilities of the neural network. To this end, two types of pruning strategies are investigated:

- Deterministic learnable masks that observe the network parameters.
- Probalistic dropout layers that omit each neuron with a specific dropout probability.

Both approaches aim to learn the network parameters with the most influence to the prediction accuracy and use sparsity inducing l0 or l1 regularizers to converge to a sparse solution [LL16].

In this study the potential of deterministic learnable masks in the form of a simple binary mask and the Smallify algorithm by Leclerc et al. [Lec+18], as well as probalistic dropout in the form of variational dropout [MAV17] are investigated. The following sections explore these algorithms in more detail.

2.4.1. Trainable Masking

A simple idea for implementing a basic pruning method was given by Rho et al. [Rho+22], who use a straight through estimator to train a binary mask on the network weights to obtain a sparse representation of their underlying network. In the case of a simple, fully connected network, such a pruning algorithm is implemented by introducing a series of additional network layers subsequent to each of the original layers. These pruning layers consist of neurons that receive the output of the previous layers as input, which they then pass on to the subsequent layer after filtering out some of the information. Filtering is done, for example, by functions that set an output to either 0 or 1, where a value of 0 deactivates the output of the neuron and a 1 passes it along to the next layer. The use of such binary neurons can be utilized to give rise to sparse representations of the original network, where the most irrelevant neurons to the optimization task are learned to be deactivated and can be pruned.

To this end, most neural networks use the backpropagation algorithm to train the internal state of the network. This is done by computing the gradient of a given loss function with respect to the network weights for an observed input-output pair. The algorithm iterates backwards through the network, applies the chain rule and calculates the gradient of each layer, given the gradient of the previous layer. According to a calibrated learning strategy, the weights of each layer are then updated, to minimize their impact on the loss. If the network contains the aforementioned binary pruning layers, then the derivatives of these layers are mostly 0 everywhere (and go to infinity if a neuron is not dropped), leading to gradients that are largely constant and flat, and thus impractical for the Backpropagation algorithm. Training of binary pruning layers without simplifications or estimations is thus not feasible.

The straight through estimator [BLC13] is a simple approximation technique that aims to estimate the gradient of a loss function with respect to the input of binary or otherwise non-smooth neurons. In the case of the simple binary pruning, the straight-through estimator is used in backpropagation and serves as an approximator for the gradients in such binary layers. This is achieved by simply ignoring the derivative of the binary function represented by the pruning layer and passing on the incoming gradient without modification, like the gradient of the identity function. Although this is a very simple and biased estimator, it is relatively easy to implement and to compute, and recent work by Le et al. [Le+22] and Rho et al. [Rho+22] note success with this

method.

The masks are trained simultaneously with the rest of the network. Thus, the network is able to learn which parameters are most important for reconstruction accuracy and can turn off most of the unimportant network parameters without significantly affecting prediction performance. To incentivise the network to learn sparse latent feature grid representations, and additional pruning loss term is introduced to the overall training loss. This is done in form of the l1 norm on the mask values of the pruning layers. Since the l1 norm acts as a sparsity constraint that pushes the mask values M to 0, an additional term for the network weights θ is added to the training loss to account for the effects of the binary pruning layers on the overall loss function:

$$Loss(x, y; \theta, M) = Loss(x, y) + \lambda_1 \cdot ||M||_1 + \lambda_2 \cdot ||\theta||_p^p \tag{2.10}$$

2.4.2. Smallify

Smallify [Lec+18] is a form of *targeted pruning* that optimizes network size and inference time during training. It uses structured pruning to remove neurons with the least contribution to prediction accuracy from an oversized network, thereby learning the optimal network size at the same time that the network optimizes for prediction performance.

Pruning is done by extending the original, unpruned network with a new layer type, the *Switch Layer*, that can switch neurons on and off. This layer-based approach to pruning makes it easy to implement Smallify in numerous neural network frameworks and existing network architectures. A Switch Layer is placed after each layer, which size is supposed to be pruned and is co-optimized with the rest of the network during training. Each Switch Layer is parameterized by a vector $\beta \in R^c$ of size c, where c is the number of neurons in the preceding layer. When the output of a Layer L passes through the subsequent Switch Layer S, the output of each neuron i is multiplied by a learnable $\beta_i \in [0, 1]$:

$$S_\beta(L(x)) = \beta_i \cdot L(x)_i, \forall i \in [1, \ldots, c] \tag{2.11}$$

The β parameters are initialized according to a normal distribution: $\beta_i \sim \mathcal{N}(0, 1)$. If $\beta_i = 0$, the $i - th$ neuron is multiplied by zero and is deactivated for any further computations after the Switch Layer, while a β_i- value of 1 will let the whole signal go through. All deactivated neurons can then be safely removed from the network, thereby shrinking network size and improving inference times.

The objective of Smallify is to maximize the number of deactivated switches to reduce the model size as much as possible while maintaining prediction accuracy. This is done by jointly training the network for it's reconstruction accuracy and the pruning layers for a high pruning rate by combining the training loss of the network with an

additional pruning loss of the Smallify method. The pruning loss applies a l1 norm to the β parameters of the Switch layer, thereby incentivising the Switch layer to deactivate neurons. To account for the scaling of the Switch Layers on the network activations during training, the authors of Smallify propose to additionally implement a weight loss that pushes the l2 norm of the network weights θ toward 0. The final loss used for optimizing the network then consists of:

$$Loss(x, y; \theta, \beta) = Loss(x, y; \beta) + \lambda_\beta \cdot ||\beta||_1 + \lambda_w \cdot ||\theta||_p^p \tag{2.12}$$

where λ_β and λ_w control the influence of the added pruning and weight losses on the optimization.

Finally, the authors of Smallify note that it is unlikely that the unimportant components of β will ever be exactly 0 and propose a different approach to the deactivation of neurons. They recognize that the l1 penalty causes irrelevant neurons to oscillate their Switch Layer close to 0, while never reaching exactly 0. To detect this case, they propose to deactivate neurons according to the *Sign Variance Strategy*: At each update the sign of each β-component (-1 or 1) is measured. Then, the exponential moving average (EMA) of its mean and variance are calculated. If the variance exceeds a predefined threshold, the neuron is considered not to contribute significantly to the output and is therefore disabled.

2.4.3. Variational Dropout

Variational dropout is proposed by Molchanov et al. [MAV17] as an effective tool for pruning networks with a high degree of dropout flexibility. The method is extended by Neklyudov et al. [Nek+17] to a structured dropout algorithm that is more suitable for pruning. Variational dropout extends dropout to *Bayesian Networks* and creates an adaptive dropout scheme. This allows the dropout algorithm to treat the dropout rate not as a preset hyperparameter, but it can be learned individually for each layer through training.

Bayesian Neural Networks are a subclass of neural networks that excel at estimating the uncertainties present in the prediction process of a neural network. This is useful when the task of the network is not limited to making a particular prediction, but also to informing the user of the network's certainty that the prediction is correct (for example when prescribing medication). In case of the network pruning process, the parameters with the most uncertainty could then be pruned, since they carry no meaningful information. Bayesian neural networks measure this uncertainty by introducing stochastic components into the network in the form of stochastic activation functions or stochastic weights. Instead of assigning static values to the parameters of the network, the parameter values get modeled after specific distributions (See

Figure 2.6) [Jos+22]. The estimated variances of these distributions can then be used as an indicator of the model's uncertainty for a given input.

Given a data set D consisting of input values $X = \{x_1, \ldots, x_n\}$ and observed values $Y = \{y_1, \ldots, y_n\}$ the goal is to find parameters Θ for a neural network f, such that $f_\Theta(X) = Y$. Following the Bayesian approach, this requires some *prior* knowledge about the distribution of network parameters $p(\Theta)$. Here, the prior represents an initial belief about which parameterizations of the network are likely or unlikely to produce the observed values.

In *Bayesian Inference*, the data set D is used to transform the prior distribution into a *posterior distribution* $p(\Theta|D) = p(D|\Theta) \cdot p(\Theta)/p(D)$, which can be used to generate the network parameters from [MAV17]. The bayesian posterior for neural networks is a complex and highly non-convex probability distribution that requires the computation of intractable multidimensional integrals to solve.

To address this problem, *variational inference* has been introduced as an approximation technique. Rather than sampling from the exact posterior, this approach uses a parameterized distribution $q_\varphi(\Theta)$. The values of the *variational parameters* φ are then learned, such that $q_\varphi(\Theta)$ closely resembles $p(\Theta|D)$ with a given initial prior $p(\Theta)$ [KSW15]. The quality of this approximation is measured with the *Kullback-Leibler divergence* $D_{KL}(q_\varphi(\Theta)||p(\Theta|D))$, which is combined with the *expected log-likelihood* to form the *variational lower bound* for optimization.

Molchanov et al. [MAV17] and Neklyudov et al. [Nek+17] implement variational dropout as a set of dropout layers that produce their output y by applying multiplicative noise Ξ on the output of a preceding layer $x \in \mathbb{R}^l$ of size l:

$$y_i = x_i \cdot \Xi_i \tag{2.13}$$

where Ξ is treated as a continuous noise variable $\Xi_i \sim \mathcal{N}(1, \alpha = \frac{p}{1-p})$ that has been proven to work similar to a binary dropout approach, where a neuron is deactivated with dropout probability p [Sri+14]. According to the variational inference approach, multiplying noise on an activation value is the same as sampling the activation value from a correspondingly parameterized Normal distribution [KSW15]. This means that Ξ can be understood as being generated by applying a randomly sampled Gaussian noise $\xi \sim \mathcal{N}(1, \alpha)$ on a mean value θ. Then Ξ is treated as a random variable parameterized by θ and α:

$$\begin{aligned} \Xi_i &= \theta_i \cdot \xi_i = \theta_i \cdot (1 + \sqrt{\alpha_i} \cdot \epsilon_i) \sim \mathcal{N}(\Xi|\theta_i, \alpha \cdot \theta_i^2) \\ \epsilon_i &\sim \mathcal{N}(0, 1) \end{aligned} \tag{2.14}$$

The noise values of the variational dropout layers are then generated by approximating the sampling of the aforementioned Normal distribution through the posterior distribution $q(\Xi|\theta, \alpha)$ and given prior $p(\Xi)$. Molchanov et al. [MAV17] define $p(\Xi)$ as a

logscale uniform distribution, that samples points uniformly between two values $log(a)$ and $log(b)$ and has been shown to have sparsification properties for neural networks:

$$p(log(|\Xi_i|)) = const \iff p(|\Xi_i|) \propto \frac{1}{|\Xi_i|} \tag{2.15}$$

The parameters θ and α are handled as variational parameters and optimized by the dropout layer. Specifically, this enables variational dropout to tune $\alpha = \frac{p}{1-p}$ for each layer, thus learning an optimal dropout rate for each layer. As the primary goal of variational dropout is to produce sparse weight matrices, the case where $\alpha >> 1$ is especially interesting, since $\alpha_i \to +\infty$ corresponds to a droprate $p = 1$ and the affected neuron can be safely pruned. According to Molchanov et al. [MAV17], large α values complicate the training of the variational dropout process, since the gradients vary greatly in this case. They propose to reduce the gradient variance by replacing the multiplicative noise term $1 + \sqrt{\alpha_i} \cdot \epsilon_i$ by an additive noise term $\sigma_i \cdot \epsilon_i$ that treats $\sigma_i^2 = \alpha_i \cdot \theta_i^2$ as a new independent dropout variable. The noise for each dropout layer can then be generated as:

$$\begin{aligned} &\Xi_i = \theta_i \cdot (1 + \sqrt{\alpha_i} \cdot \epsilon_i) = \theta_i + \sigma_i \cdot \epsilon_i \\ &\epsilon_i \sim \mathcal{N}(0, 1) \end{aligned} \tag{2.16}$$

During the training of a network, the goal is to maximize the structural sparsity of the network architecture while maintaining the predictive accuracy of the network. Given a data set D of N output-input pairs $(y_n, x_n)_{n=1}^N$, this is done by substituting the original optimization loss of the network architecture with the variational lower bound $L(\Theta)$ to simultaneously train the prediction accuracy and sparsity of the network:

$$L(\Theta) = L_D(y|\mu, psigma) - D_{KL}(q(\Xi|\theta, \alpha)||p(\Xi)) \to \max_{\Theta} \tag{2.17}$$

When applying variational dropout to a network, the predictions of the network have to be understood as probabilistic rather than deterministic. This means that the network no longer generates deterministic scalar values. It instead uses a network prediction μ and a predicted or given variance $psigma$ to define a posterior distribution $\mathcal{N}(\mu, psigma)$ over the ground truth data, which is then used in the optimization. $L_D(y|\mu, psigma)$ describes the expected log likelihood. In this stochastic environment it is the primary substitute for the original ground truth network loss and is responsible for aligning the predictions μ of the network with the ground truth data. If the input x_n is fed into the network and transformed by the network's fully connected and dropout layers into a prediction μ, the log likelihood is defined as follows:

$$L_D(y_n|\mu, psigma) = -(y_n - \mu)^2 / (2 \cdot psigma^2) - \log(\sqrt{2 \cdot \pi \cdot psigma^2}) \tag{2.18}$$

This parameterization allows the sensitivity of the log-likelihood loss to the ground truth data to be tuned using the parameter *psigma*, which is either statically fixed or predicted by a second network.

The Kullback-Leibner divergence $D_{KL}(q(\Xi|\theta,\alpha)||p(\Xi)))$ acts as a regularization term that trains the approximate posterior distribution $q(\Xi|\theta,\alpha)$ of the dropout noise to follow the assumed prior $p(\Theta)$. The D_{KL} can be calculated by decomposing the divergence term into a sum for each dropout layer i:

$$\begin{aligned} &D_{KL}(q(\Xi|\theta,\alpha)||p(\Xi))) = \\ &\sum_i D_{KL}(q(\Xi_i|\theta_i,\alpha_i)||p(\Xi_i)) \end{aligned} \tag{2.19}$$

For the chosen approximate posterior $q(\Xi|\theta,\alpha) = \mathcal{N}(\Xi|\theta_i, \alpha \cdot \theta_i^2)$ and prior $p(\Xi)$, the D_{KL} divergence of each dropout layer cannot be computed analytically [KSW15], but can be accurately approximated. Molchanov et al. [MAV17] propose to choose an approximation that includes the sigmoid function $\sigma(\cdot)$ and numerous constants to produce a function that behaves similarly to the true negative D_{KL} divergence:

$$\begin{aligned} &-D_{KL}(q(\Xi_i|\theta_i,\alpha_i)||p(\Xi_i)) \approx \\ &k_1\sigma(k_2 + k_3 log(\alpha_i)) - 0.5 log(1+\alpha_i^{-1}) + C \\ &k_1 = 0.63576\ k_2 = 1.87320\ k_3 = 1.48695 \end{aligned} \tag{2.20}$$

Since the prior defines the sparsifying properties of the variational dropout approach, the D_{KL} influences the sparsity of the final network configuration by favoring large values of α.

In order to affect the prediction accuracy as little as possible with the pruning of the neurons, one needs to make sure that the weights of neurons corresponding to dropout nodes with high α values are close to 0. For this purpose, an additional weight-loss is introduced to the variational lower bound that ensures that the layer weights with noisy dropout input are shifted to 0:

$$\min_{\Theta} L_D(\Theta) + \lambda_D \cdot D_{KL} + \lambda_w \cdot ||\Theta||_2^2 \tag{2.21}$$

where λ_D and λ_w are scaling terms for the D_{KL} and weight loss fractions during optimization.

Pruning is performed after training by calculating the α and dropout values for each droppable neuron in the network and culling all neurons whose drop rate exceeds a certain threshold. Then the weights of each layer are multiplied with the corresponding θ parameters of the dropout layers and the dropout layers are discarded, in order to not introduce an overhead of parameters to the network.

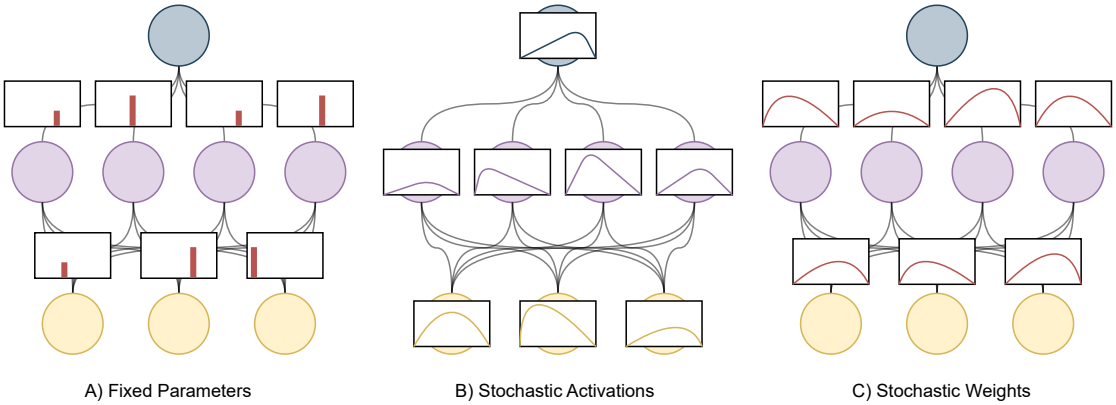


Figure 2.6.: (a) Traditional neural network with fixed values for the parameters, (b) stochastic neural network with a probability distribution for the activations, and (c) stochastic neural network with a probability distribution over the weights. Adapted from [Jos+22].

2.5. Neural Architecture Search

Although neural networks are powerful tools for a variety of challenges, their success depends heavily on the selection of appropriate hyperparameters by the user. Hyperparameters determine the network architecture (e.g., by specifying the number or size of hidden layers) and the training process (e.g., by deciding how much data to use in each training iteration or how fast to update the network's parameters). As such, these types of variables are not learned by the network itself, but are fixed before training. The goal of Neural Architecture Search (NAS) is to reduce human interaction as much as possible when searching for optimal *hyperparameters* of a neural network. The fundamental concept is to generate these parameters automatically, thereby ensuring the best performance of the underlying network without relying on the researcher's bias or prior knowledge [Ren+21].

The task of Neural Architecture Search is closely interlinked with the task of multi-objective optimization, which aims to maximize multiple defined objectives, given a set of changeable parameters. In this case the NAS algorithm aims to maximize the potential of the network with respect to a defined objective (e.g. generating a shallow network with high prediction accuracy), while influencing the network by changing the hyperparameters. In most cases, these problems possess no single best solution. Rather, the goal is to compute a *pareto frontier*: a set of optimal trade-offs where the improvement of one objective means the deterioration of another. Using the calculated pareto frontier, an individual decision maker can then choose an objective trade off and

specific parameters according to their preferences [Eri+21].

Early work in this domain includes MetaQNN by Baker et al. [Bak+16], which uses reinforcement learning to automatically generate high performing network architectures. Reinforcement learning is based on a reward function, which defines a certain goal and assigns reward values to certain states or actions. This enables a neural network to autonomously learn different strategies for different situations to maximize the given reward. By defining a finite and discrete search space for the network parameters, as well as specifying the reward function to encompass a goal for multi-objective optimization, this method can be used for Neural Architecture Search. In this case, the network alternates between phases of exploration, in which it learns about its search space through random sampling, and exploitation, in which it uses its prior knowledge to select more powerful models.

Large-Scale Evolution by Real et al [Rea+17] is a another widely used method for Neural Architecture Search. The method treats NAS as a kind of tournament where model architectures of high quality win over lower ones. This is done by managing a population of trained architectures and assigning a quality measure to each model based on model accuracy or other user-defined characteristics. In each search step two models from the population are compared at random and the architecture with lower quality is removed from the model pool. The winning model is then selected as the basis for a newly created model, where a mutation strategy changes the parameter of the parent model and applies them to the child model. The child model is then trained, evaluated, and put in the population to act as a parent model in the next search step.

While both reinforcement learning approaches and evolutionary algorithms are popular choices for solving multi-objective optimization, both methods suffer from a high sample complexity, making them infeasible for optimizing functions with large parameter search spaces that are expansive to evaluate.

In this work AX [Bak+18] is used to run multi-objective NAS for parameterizing learning-based compression algorithms. In order to find the best tradeoffs between multiple objectives of interest, e.g. maximizing model accuracy while minimizing model size, AX uses a *bayesian optimization* strategy. Bayesian optimization treats the optimization of neural networks as a *black-box optimization* problem, where the network $f_\Theta(x)$ is a black box without any insight about analytical expressions of f or its derivatives. Thus, evaluation of f is restricted to sampling and feeding a point x into the network and determining a possibly noisy (e.g., by measurement error or randomness in network training) response. Bayesian optimization is a model-based optimization method that incorporates a prior belief about f and updates the prior during optimization to generate a posterior that better approximates f. The model used to approximate f is called the *surrogate model*, while the *acquisition function* decides which parameters in the search-space should be evaluated next. While

evaluating the true black-box function can be time-consuming or costly, evaluating the surrogate is inexpensive, relatively fast, and parallelizable, which is why bayesian optimization has established itself as a sample-efficient method for searching neural architectures [Kan+18].

The work of Eriksson et al. [Eri+21] extends this method by fusing bayesian optimization with the acquisition function for parallel noisy expected hypervolume improvement (qNEHVI)[DBB21]. The implementation of qNEHVI on top of bayesian optimization allows for parallel evaluation of multiple architectures and naturally smoothes the observational noise present in both the latency and accuracy metrics. This allows the experiments conducted in this thesis to perform efficient and parallel multi-objective optimization and thus provides each experiment with the best network parameterizations of the different network architectures under investigation.

3. Related Work

This chapter discusses the current state of research on Scene Representation Networks and network pruning algorithms, and provides justification for the selection of specific model architectures and network compression algorithms examined in this work.

3.1. Scene Representation Networks

Traditional graphics representations, such as meshes, point clouds, and volume data, are widely used as reliable all-purpose tools in a variety of applications, from visual effects and computer games to three-dimensional computer vision. Although these traditional representations are well researched, they have their own drawbacks and may not be suitable for new emerging applications in neural rendering, imaging, and simulation. Modern research in these areas often needs data of high resolution that is scalable in learning-based pipelines and as such end-to-end differentiable and quickly optimizable. Conventional representations often conflict with these properties because their accuracy scales according to the Nyquist sampling criterion and necessitates the explicit storage of spatiotemporal samples in memory.

A wide variety of Scene Representation Networks (SRN) have been developed in the past years to create new means of data representation that overcome these limitations. SRN are a class of fully connected, often shallow neural networks that encode continuous signals of arbitrary dimension. For example, let F be a multi parameter field that maps to each point in a given domain a set of D parameters. SRN implicitly encode F as a neural network f comprised of multiple fully-connected layers. Usually, the network takes a spatial domain position p as input and predicts the color or density at that position in the original field: $f_\Theta : \mathbb{R}^3 \rightarrow \mathbb{R}^D$. Neural scene representation techniques are powerful and widely applicable to problems in visual computing and beyond. They provide the ability to directly reconstruct individual samples of the original data at any resolution, do not rely on specific structures of the underlying data to function correctly, and can efficiently compress non-local coherence in the data. Apart from the potential of SRN as an alternative signal storage format, they are also widely used in related areas of computer vision, such as novel view synthesis and few-shot scene reconstruction, object manipulation and interpolation, compression or robotics.

Early work in the domain of signal reconstruction focuses on encoding the underlying data as an implicit function that is implemented as a fully connected deep neural network. In this manner DeepSDF by Parket et al. [Par+19] and IM-NET by Chen and Zhang [CZ19] present a method for extracting features from a surface geometry that represents the original surface by its continuous signed distance function. At rendering time the network is then fed with spatial positions p and tasked with predicting the SDF value at position p, so that the original surface can be extracted as an iso-surface at arbitrary resolution. Apart from signal representation, the networks also provide the ability to interpolate or complete a signal from partial or noisy data. This is done by concatenating p with a latent code vector z that encodes the desired shape of the represented surface. While IM-NET uses an *encoder-decoder* structure for training and inference, DeepSDF employs an *auto-decoder* architecture that consists of a set of neural network layers. Contrary to the encoder-decoder structure, where a descriptive latent feature vector is calculated from an encoder network, the auto-decoder initializes random latent vectors for each data point at the beginning of training and optimizes the latent vectors and decoder weights during training with back-propagation. The resulting latent feature space can be interpolated and results in a model that can represent a variety of different shapes with one training, by interpolating or switching the optimized latent vectors. Shape completion can be achieved by fixing the decoder weights during inference and optimizing the input latent feature code for the lowest error in the learned latent feature space.

Alternatively, Occupancy Networks by Mescheder et al. [Mes+19] propose to represent 3D geometry as a learned occupancy function rather than an implicit signed distance function. They argue that an efficient way to represent surface geometry would be to learn a continuous three-dimensional occupancy function that predicts if the surface exists at the queried points in space. An occupancy network is defined as a deep neural network that takes spatial input position p and an observation x as input outputs a real number between 0 and 1, which represents the probability of occupancy of the surface at point p.

Acorn by Martel et al. [Mar+21] builds upon the occupancy-networks by introducing adaptive data structures that can be optimized during training to allocate more resources in areas of fine detail and great interest. They reason that conventional data display formats are not capable of displaying scenes with high resolution or large scenes with thousands of polygons. While explicit data representation formats are fast to evaluate, they also require storing a large number of explicit features and thus scale poorly with data resolution or complexity in regard to memory requirements. Implicit data representations in the form of learned neural networks are more memory efficient, but struggle to find applications in real-time systems, since their representation accuracy is often related to their internal network complexity and thus require

large and slow forward passes for data evaluation. Martel et al. tackle this problem by suggesting a two-stage network design. In the first stage, a large encoder network divides the domain of the input signal into a multiscale data structure, similar to an octree. The data structure represents a local feature grid of the input data, where regions of interest are represented with a higher resolution. In the second stage, spatial positions are passed to a smaller decoder network for evaluation, which interpolates the feature grid at the given positions and outputs an occupancy value. Since the majority of the network parameters reside in the encoder network and the decoder network consists of only a small fully connected model, the Acorn architecture significantly reduces the computational effort required to evaluate the underlying data. Only a single large forward pass is required to build the adaptive feature grid and subsequent data evaluation is handled with fast forward passes of the small decoder network.

Müller et al. [Mül+22] extend on the idea of adaptive data structures with spatial hashing. They reason that the use of a multiresolution hash-encoding of the encoded signal's feature space can lead to an efficient, widely applicable and easily parellelizable representation, in contrast to the more task-specific data structures used in Acorn [Mar+21] or by Takikawa et al. [Tak+21]. The hash-encoding uses a feature space that is arranged into grids of different levels, with each level containing multiple feature vectors. The grids of each level are independent to one another and store feature vectors at the vertices of each grid. The feature vectors themselves are stored in an array for each layer, and a hashing function performs the mapping of each grid point to its representative array entry. At coarse resolutions, there are a similar number of grid points and available array entries, so the hashing function performs a simple 1:1 mapping. At finer resolutions, the number of grid points exceeds the number of available entries in the feature vector array and the array is treated as a hash table to resolve collisions. During training, the network learns to favor the feature vectors with the greatest relevance to the training loss, and is automatically incentivised to represent areas at different resolutions with their most important details. During data evaluation, a spatial input position p is fed into the network and assessed at L resolution levels. The surrounding grid points at each level are then calculated and the corresponding feature vectors are looked up by the hash function. Subsequently, the resulting feature vectors are interpolated according to the relative position of p in the grid for each level. Lastly, the interpolated feature vector is concatenated and passed to a small network that reconstructs the original data.

In a different way, NeRF [Mil+20] extends the idea of simply reconstructing learned geometries by proposing a method for generating novel views from 2D input images without the help of additional latent feature information. This is done by combining direct volume rendering with a neural network to represent static scenes as a learned

5D function

$$f_\Theta(p, v) \rightarrow (c, \sigma) \tag{3.1}$$

where $p = (x, y, z)$ represents an in-scene position, $v = (v_1, v_2)$ represents the viewing direction, $c = (r, g, b)$ represents the radiance and σ represents the density. The network is split into two stages. In the first stage a fully connected network converts the input p into a prediction for σ and a high-dimensional feature vector. In the second stage this feature vector is then concatenated with the viewing direction v and fed into a second network that predicts the output radiance c.

In order to generate novel views, rays $r(t)$ are traced through each pixel of the to-be-synthesized image and the network is probed along the rays. Direct volume rendering techniques are then used to combine the generated density and radiation values of the network into photorealistic colors $C(r)$ of the new image. The network is trained on a set of 2D input images with known camera poses that are used for optimization by minimizing the square error between the predicted color $C(r)$ and the ground truth $C_{gt}(r)$ (See Figure 3.1).

The high visual accuracy as well as the easy-to-implement formulation of NeRF has since inspired a large collection of follow-on work focused on further extending the quality and real-time capabilities of NeRF. Improvements include enabling NeRF to be trained from fewer input views ([Den+22], [Yu+21], [Che+21]) or significantly speeding up the rendering process of the network ([Den+22], [Gar+21]).

3.1.1. Compressive Scene Representation Networks

This work is primarily interested in the capabilities of learning-based compression algorithms on scientific volumetric data. Unlike the memory required for discrete parameterizations (such as 3D meshes, point clouds, or voxel grids) which scale poorly with larger resolution, the memory required for neural scene representations instead scales with the complexity of the network, i.e., with the number of parameters required.

A closely related approach to such data compression is spatial *upsampling*. In the context of neural networks, upsampling (*Super-resolution*) are techniques that aim to enhance the image, video, or volumetric data resolution from initial low-resolution data with the help of predictive deep neural networks. Since only the trained network model and the low-resolution data need to be stored for complete data recovery, this method brings the immediate advantage of saving storage space by reducing data. In addition, super-resolution methods provide the ability to approach the underlying data at different levels of detail, which greatly decreases computational complexity if the computational task allows for a tolerable error.

Zhou et al. [Zho+17] realise the potential of SRN for volume upscaling and propose a network that directly learns a continuous mapping from low-resolution blocks to high-resolution volumes. Wurster et al. [Wur+21] extend this work by combining Super-resolution neural networks with the resource efficiency of hierarchical data formats. Their approach adds flexibility to upscale input data with arbitrary levels of detail, while minimizing scaling artifacts. Upsampling is done by creating a network architecture that takes volumetric data represented by an octree data structure as input and outputs a corresponding high-resolution uniform grid. The architecture consists of a hierarchy of neural networks, where each network in the hierarchy is responsible for upscaling one level of detail to the next resolution. Artifacts are avoided by performing upsampling for the entire data area instead of separately upscaling individual data sections with different levels of detail.

While super-resolution techniques can achieve good compression ratios by regenerating high-resolution data from sufficiently down-sampled low-resolution data, these methods are not specifically designed for compression and may be outperformed in compression ratio and real-time applicability by alternative compressive networks that implicitly represent the input signal and are able to exploit non-local coherence in the data.

TINC by Yang et al [Yan+22] leverages a hierarchical parameter sharing mechanism to achieve remarkable compression results even for large scene data. They propose to divide an input scene into local regions by octree partitioning, which can then be compactly represented by implicit networks. The individual networks are then organized in a tree structure and share their features in a hierarchical manner according to the spatial distance of their encoded region. The hierarchical tree structure of the neural networks allows higher-level nodes to extract increasingly global information from their lower-level nodes, while providing a detailed but compressed representation of input data through the parameter sharing method.

Although the use of adaptive data structures achieve considerable results in the domain of data representation and data compression, the use of these structures assumes the base data to have large regions of low frequency details, that can be efficiently represented by the adaptive resolutions of the encoding data structure.

This work aims to analyze and improve implicit compression techniques that make no assumptions about the structure of the original data and excel at data compression tasks by capturing both global and local features of the input data, thereby succeeding for a wide range of use cases. In particular, studies in this thesis are interested in the following two network architectures:

- Neurcomp by Lu et al. [Lu+21] proposes a compression scheme based on overfitting a deep neural network directly to arbitrary scalar input data. By limiting

the amount of available parameters to less than the original data size, the network itself functions as an implicit compressed version of the data. As a result, only the parameters and architecture of the learned network have to be stored, instead of the original data. The original work additionally employs quantization techniques to further compress the network parameters, thus creating a learning based compression technique that achieves remarkable compression ratios while still preserving important features of the input volume. This study explores the potential of using pruning techniques on the network parameters to enhance compression by sparsifying the network parameters prior to quantization.

- In contrast, fv-SRN by Weiss et al. [WHW22] trains a small network similar to NeRF but extended with a low-dimensional latent code vector to represent the original data. The small size of the network makes it possible to load all network parameters into the shared memory space of a GPU, thereby enabling the efficient use of GPU tensor cores and on-chip raytracing kernels to significantly speeding up the reconstruction task. Since most of the memory requirements are concentrated on the latent code grid, this thesis aims to find efficient methods to compress the grid further using pruning algorithms and wavelet transformations.

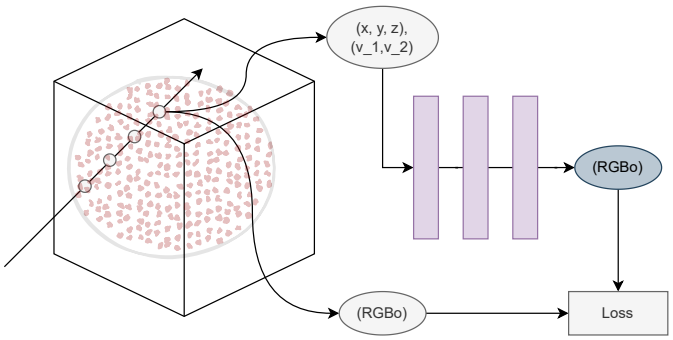


Figure 3.1.: The NeRF rendering process. The algorithm shoots rays through each pixel of the final image and evaluates the network along the rays. The generated color and opacity values are then combined in order to render the final image. Adapted from [Gao+22].

3.2. Compression of Deep Neural Networks

Modern neural networks, as discussed in section 2.4, depend on deep architectures consisting of millions of parameters in order to achieve high prediction accuracy, and

thus require efficient compression techniques for usage in small, portable systems or real-time applications. Most modern neural network compression algorithms can be generalized into four categories [Che+17]: *low-rank factorization*, *transferred/compact convolutional filters*,*knowledge distillation* and *parameter pruning and quantization*.

Low-rank factorization based techniques use matrix or tensor decomposition to estimate the most informative parameters of the neural network. Algorithms that use this approach are TTHRESH [BLP19] and Wavelet Decomposition, described in subsection 2.2.1 and subsection 2.2.3. Work by Sainath et al. [Sai+13] apply this concept specifically on deep neural networks and use a low-rank matrix factorization on the weight matrix of the final layer of the neural network. They assume that the last layer represents a low-rank weight matrix that can be factorized to be represented by two smaller matrices, thereby reducing the number of parameters prior to training. Although low-rank factorization is a straightforward method that yields good results in model compression and acceleration, the associated decomposition operations are often computationally intensive and require specialized hardware for effective use. Moreover, these approaches require that the underlying data admit a low-rank decomposition in the first place, e.g., by consisting largely of low frequencies in the case of Fourier bases.

Next, transferred/compact convolutional filter based approaches design special structural convolutional filters to reduce the parameter space and save memory and computational cost. GhostNet by Han et al. [Han+20] leverage this idea by proposing a method that exploits existing redundancy in intermediate feature maps of a CNN and thus reduces the required resources. Instead of applying large and computationally intensive convolutional filters to the underlying data to extract the required feature maps, they propose to use a set of smaller intrinsic feature maps that can be linearly transformed to produce larger and more powerful feature maps. While these transform-based parameter sharing techniques work well for large architectures with wide filters, they do not yet see success in thinner model architectures and are restricted to convolutional neural networks.

Alternatively, knowledge distillation based methods first learn a large model and subsequently train a more compact neural network to reproduce the output of the larger network in a student-teacher relationship. Work by Hinton et al. [HVD15] and Fitnets by Romero et al. [Rom+14] are examples of this technique. They propose to train thin and deep neural networks to compress more complex but shallower networks. The thin compressive networks are trained by following a student-teacher paradigm, in which the student is penalized according to the complex teacher's output and learns to mimic the feature maps of the teacher. Even though the concept of knowledge distillation is interesting and succeeds in reducing the computational cost and memory requirements of deeper models, they have the disadvantage that they can only be applied to tasks described with a softmax loss function and generally produce less competitive results

compared to other compression methods.

On another hand, quantization approaches (as described in subsection 2.2.2) compress an original network by reducing the number of bits required to represent each parameter. A straightforward approach is used by Gong et al. [Gon+14], as well as Neurcomp [Lu+21], who use k-means clustering to quantize the network weights. Other know works that effectively use quantization include Deep Compression by Han et al. [HMD15], who use codebooks to quantize network parameters into bins that are described by their bin-centroids and subsequently refine the codebooks and centroids to best reconstruct the original data. All in all, quantization is an easy-to-implement solution that is often combined with other algorithms for additional data compression.

Lastly, parameter pruning and dropout based methods examine the redundancy in the model parameters and attempt to remove the redundant and non-critical parameters. This can either be done by introducing metrics that observe the network during training or probing the network parameters after the learning task has finished.

This work primarily analyzes the impact of pruning on the compressibility of neural networks. Pruning is effective and has been proven to work for a wide variety of network architectures [Bla+20], while also making no assumptions about network structure, and being relatively easy to implement for an already given network architecture. The following section provides a broader overview of the topic of neural network pruning.

3.2.1. Neural Network Pruning

A pruning algorithm aims to prune redundant, non-informative parameters in a DNN model. Pruning can be used to achieve many different goals, such as reducing the storage footprint or the computational complexity of inference for a neural network.

Examples of popular pruning algorithms include grouping parameters into hash buckets for parameter sharing [Che+15], as well as deep compression [HMD15], which prunes insignificant parameter connections after training, together with *parameter quantization* and *Huffman encoding* to compress the neural network. There is also growing research interest in training compact DNN with sparsity constraints, so that the network is already pruned during the learning process. One way to achieve sparsity during training is to enrich the loss term with l0 or l1 regularizers, thereby allowing the optimization problem to converge to a sparse solution [LL16]. A popular form of pruning with the explicit goal of network compression while preserving prediction quality is *targeted pruning* [Gom+19]. In this dropout method neurons are adaptively selected according to a parameter ranking to drop out in such a way that the network adapts to the neuronal pruning, allowing it to be significantly reduced in size without much loss of accuracy. Smallify by Leclerc et al. [Lec+18] leverages this approach

by introducing a *Switch Layer* that simultaneously optimizes network size and model performance with structured pruning during training.

Apart from network weights and biases, Rho et al. [Rho+22] propose to extend pruning methods to networks that leverage latent feature grids in combination with fully connected neural networks. They employ an element-wise binary mask to increase the amount of zero-elements in the grid. The mask is optimized along with the network and grid parameters to learn an appropriate dropout probability for each grid element that zeroes out the majority of the grid coefficients without significantly decreasing the prediction accuracy of the network architecture.

Generally, pruning layers can be considered as one of two types: Pruning methods that prune individual parameters of the network weight matrices are labeled as *unstructured pruning*, while algorithms that prune parameters in groups to remove entire neurons from the model are termed *structured pruning* (See Figure 3.2). Employing unstructured pruning results in a sparse network with a smaller parameter count, but the resulting sparse weight matrices may not be exploited for speedup or smaller memory footprint by modern hardware. Structured pruning, on the other hand, removes entire neurons or channels from the network, which reduces the overall size of the weight matrices in the network, and can thus be utilized by hardware and software for dense computations.

Regardless of the goal, pruning is a tradeoff between model efficiency and quality, where pruning increases model efficiency while quality decreases.

Rather than using deterministic learnable layers to prune portions of the network, recent research has found success in basing pruning algorithms on probabilistic dropout layers. Dropout was introduced in 2012 as a technique to avoid overfitting. The high number of parameters in DNN makes them particularly susceptible to overfitting, since their high number of parameters teaches them to accurately represent training data but not to predict arbitrary test data. The original standard dropout method [Hin+12] omitted each neuron in a neural network with a probability of 0.5 at each training iteration, whereas testing included all neurons. This technique has been shown to significantly improve test accuracy because the network cannot access the full range of its parameters during training and is therefore averse to specializing only in training data. Mathematically, the output during training for the original standard dropout can be described by:

$$y = \phi(Wx) \cdot m, m_i \sim Bernoulli(1 - p) \tag{3.2}$$

where y is the layer output, $\phi(\cdot)$ is the activation function, W is the layer weight matrix, x is the layer input and m is the layer dropout mask with each m_i being 0 with probability p [LSV19]. After training the neuron outputs are multiplied with their

respective dropout masks and the full network is used:

$$y = (1 - p) \cdot \phi(Wx) \tag{3.3}$$

Follow up work by Srivastava et al. [Sri+14] shows that the behaviour of the standard dropout mask generated from a Bernoulli-distribution can also be replicated using a continuous distribution with the same expectation and variance. As such, multiplying neural network layers with discrete Bernoulli-noise is equal as multiplying the layers with noise generated by, for example, a Gaussian distribution $\mathcal{N}(1, \alpha = \frac{1}{1-p})$. Fast Dropout (*Gaussian Dropout*) by Wang and Manning [WM13] complements this approach and shows that the output of a layer that has undergone dropout can be computed directly by sampling from a representative Gaussian distribution. This scheme allows for significantly faster training than the standard dropout, since the sampling of the approximative Gaussian distribution eliminates the need for complex matrix multiplications of the network weights.

Although dropout methods were originally used to avoid overfitting, they are now also researched for their sparsifying characteristics, which can be used to prune and compress neural networks. According to Srivastava et al [Sri+14], standard dropout promotes sparsity in neural network weights by increasing the proportion of weights that are close to zero. As a result, dropout techniques can be used to compress neural network models by lowering and pruning the number of network parameters required for efficient operation.

Popular dropout algorithms include Ising-dropout by Salehinejad and Valaee [SV19], which applies a graphical Ising-model to a neural network to detect and drop the least useful neurons. Ising-models are widely used in statistical physics and connect a set of points in an n-dimensional periodic lattice, where each node of the lattice is assigned a binary Ising-weight (either $+1$ or -1). Two nodes are considered aligned, if they both have the same weight, and misaligned, if the weights differ. The system in the Ising model tries to reach a state in which as many nodes as possible form an aligned state. Salehinejad and Valaee try to solve the dropout optimization by mapping the activation values of the neurons to the Ising-weights in an Ising-model and use an optimizer to solve the weights in such a way that the cost of binary connections is minimized.

In addition to optimizing prediction accuracy for the complexity of a given network, dropout algorithms can also be used to generate reliable confidence intervals for network predictions. Monte Carlo dropout [Gal+16] has been introduced as an easy to implement, analogous method to produce model uncertainty estimates. This is achieved by performing a grid search on the dropout rate hyperparameter. Specifically, the network is run a number of times with standard dropout, each time with the same input and with a different randomly generated dropout mask. Each trained network can now be treated as a Monte Carlo sample, from which a Bayesian approximation

of the model space and uncertainty can be calculated. While knowledge about the certainty of a model's output is useful (e.g., for judging if a model has overfitted to its training data), the use of a grid search is often unfeasible for large, deeper models.

As an alternative stochastic method, *Variational dropout* has been shown to sparsify both fully connected and convolutional layers [MAV17], [KSW15]. Variational dropout uses a Bayesian approach similar to the Gaussian multiplicative noise presented by Srivastava et al. [Sri+14]. This method imposes a sparsity inducing prior distribution over the network weights and uses *variational inference* to infer an approximate posterior distribution for prediction instead of only point estimates. The derived adaptive dropout scheme does not treat the dropout probability p as a hyperparameter, but can automatically optimize an effective dropout probability for an entire network, individual layers or neurons. By combining dropout with a Bayesian approach, this algorithm is able to detect neurons with a high degree of uncertainty in the network, and is effectively able to prune them. The method outperforms the standard dropout scheme and achieves a large reduction in the number of parameters while minimally affecting performance. A follow up work by Neklyudovet al. [Nek+17] uses a modified *variational dropout* scheme that promotes structured sparsity, which specifically benefits the compression approach of variational dropout. This first wave of research assumes the approximate priors to follow fully factorized distributions that resemble gaussian kernels, which in turn benefits computational tractability and several optimizations with stochastic gradient-based methods for training. However, work by Nguyen et al. [Ngu+21] and Hron et al. [HMG18] demonstrates that this prior disregards the strong statistical dependencies among random weights of neural nets, making it impossible to capture the whole structure of the true posterior and to estimate the true model uncertainty. To get around the restriction of improper priors and ill-posed true posteriors many recent works in the field of variational dropout propose to employ posterior approximations with richer expressiveness. Work by Louizos and Welling [LW16] approximates the true posterior based on the parameterization of a matrix variate Gaussian distribution as a distribution over random matrices. Rather than treating each entry of a weight matrix independently, this approach considers the matrix as a whole, introducing correlations and information sharing between weights that allow for easier estimation of the true posterior value. Other research employs low-rank approximation techniques for representing the Gaussian posterior (Tomczak et al. [TST20]) or uses a prior hierarchy to obtain a joint approximation for the Dropout posterior (Nguyen et al. [Ngu+21]).

This thesis is interested in improving existing compressive Scene Representation Networks using sparsity inducing pruning and dropout algorithms. To this end the behaviour and compression quality of a selection of different pruning algorithms are

compared. First, a binary mask trained with a *straight through estimator* as a basis for comparison with more sophisticated pruning methods is examined. For more advanced pruning methods Smallify is studied as a representative for deterministic, targeted pruning, which uses trainable beta-masks together with an advanced dropout criterion to learn optimal network sizes during training. Finally, the basic implementation of variational dropout proposed by Molchanov et al. [MAV17] and Neklyudov et al. [Nek+17] is examined as a relatively easy to implement representative for the Bayesian dropout approaches, which shine in their adaptive handling of the dropout rates for each network layer.

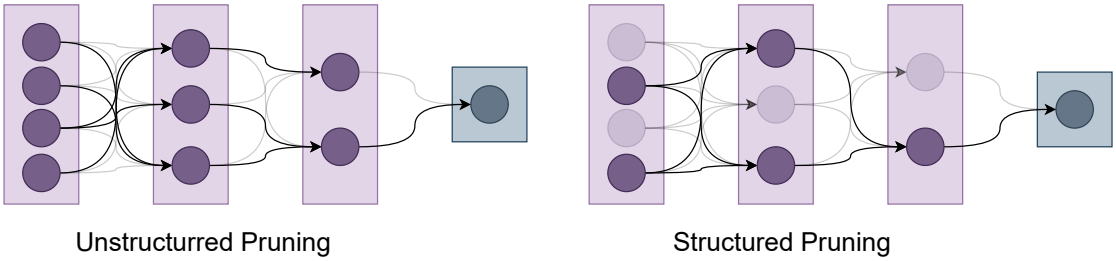


Figure 3.2.: Example of a DNN that has been pruned unstructured (left) and structured (right). The fading nodes and connections are pruned. While unstructured pruning only culls singular weight connections between the neurons, structured pruning removes whole neurons and all associated weights from the network.

4. Implementation Details

The goal of this thesis is to investigate the effects of different pruning methods on different types of compressive scene representation networks. A listing of all used hyperparameters is presented in Table 4.1. Code for the implemented methods is provided on `https://github.com/Bussler/NeurcompCompression` and `https://github.com/Bussler/Latent_Feature_Grid_Compression`. In this chapter, the concretely implemented methods are discussed in more detail.

4.1. Neurcomp

Initial experiments investigate the influence of pruning techniques on large fully connected scene representation networks. The base network architecture without additional pruning layers is implemented according to the state-of-the art architecture of Neurcomp by Lu et al. [Lu+21]. As described in subsection 2.3.1, Neurcomp is a neural network that implicitly represents and compresses voluminous input scene data. The network architecture is implemented in a broad and dynamic way to allow the user to easily evaluate many different architectures. In this context, the network architecture is also not specified to necessarily include residual blocks, but offers to investigate the effects of pruning algorithms on the compression ratio and prediction accuracy of Neurcomp with and without residual blocks. In case the residual blocks are used in the network architecture, the layer formulation in Equation 2.4 is replaced by the formulation in Equation 2.5.

Following the approach of Neurcomp, data compression is obtained by limiting the total amount of parameters m in the network to be smaller than the resolution C of the original represented data. This is achieved in one of two ways: Either the user can specify the desired number of layers and the compression ratio, and the size of the hidden layers will be calculated automatically, or the user must specify the number of layers and the layer size, which will dictate the compression ratio. In any case, given a dataset of dimensionality d, the final network f_Θ consists of a set of SIREN layers and a final layer without activation function. The first layer has a weight matrix of size $k \cdot d$, the weight matrices of the hidden layers are of size $k \cdot k$ and the last layer is of size $k \cdot 1$. The size of the bias vectors are derived in a similar way. Training is done by randomly sampling from the given data set and feeding the data into the network.

	Neurcomp	FV-SRN
Parameters	• layer amount • layer size • maximum passes • learning rate • gradient loss weighting • learning rate decay • batch size	• layer amount • layer size • grid size • feature size • maximum passes • learning rate • learning rate decay • batch size • embedding type • wavelet filter

(a)

Binary Mask	Smallify	Variational Dropout
• pruning loss weighting • weight loss weighting • pruning threshold	• pruning loss weighting • weight loss weighting • pruning threshold • EMA momentum	• psigma • D_{KL} ramp up • weight loss weighting • Entropy weighting • pruning threshold • initial droprate

(b)

Table 4.1.: Listing of available hyperparameters for (a) the examined network architectures and (b) the pruning algorithms.

Network parameters are optimized with the ADAM [KB14] algorithm, which the user can influence by specifying an initial learning rate or adjusting a learning rate strategy.

4.1.1. Pruning Methods

This section explores the implementation of the pruning methods Smallify (see subsection 2.4.2) and variational dropout (see subsection 2.4.3) in conjunction with the implemented Neurcomp architecture. The goal of adding additional the pruning methods on top of Neurcomp is to identify and remove neurons with small influence to the final prediction accuracy from the network, thereby optimizing the quality-compression ratio of the model.

When residual blocks are added to the Neurcomp architecture, the pruning layers are added to the middle layer of the residual block, since the inputs and outputs of the blocks have to be the same size, in order for the concatenation of new and older features to work correctly. In case the residual blocks are left out of the network architecture, the pruning layers can be added after each layer. This approach enables broader pruning of the network, but at the cost of reduced prediction accuracy due to the additional pruning and loss of residual block stability.

The Smallify and variational dropout layers are trained simultaneously with the basic network. After training the introduced pruning layers are culled from the network, in order to not influence the total parameter count and compression ratio. To account for the dropout effect during training, the layer weights are multiplied with their corresponding pruning layers for inference. Depending on the specific pruning strategy, the insignificant neurons of the network layers are identified and pruned as well, resulting in smaller, dense weight matrices in the network.

In order to further increase the reconstruction accuracy of the network, a finetuning strategy is employed. To this end the maximal amount of training passes over the training data is divided. Two thirds of the passes are used for simultaneous training of the network and the pruning layers. After the training has concluded, the pruning layers and insignificant neurons are culled and the last one third of passes are used to allow the network to become familiar with its pruned state.

Smallify

The implementation of Smallify is based on the pruning algorithm proposed by Leclerc et al. [Lec+18] as described in subsection 2.4.2.

Given a layer in the network with output size c, the method is added to the base network in the form of a switch layer that monitors a vector of $\beta = [\beta_1, \ldots, \beta_c]$ parameters. The layer takes the activations of a previous layer as input and multiplies

each neuron output by a parameter $\beta_i \in [0, 1]$. Neurons with a beta value of 0 do not contribute to the reconstruction effort of the network and can be effectively removed, thereby reducing the network size. In contrast to the algorithm suggested by Leclerc et al. pruning of the network parameters is performed only after the training is completed. This is based on the goal to allow the network to reactivate previously deactivated neurons in the event that they exert a noteworthy influence on the network's performance, rather than prioritizing enhancements in training time and complexity by immediately discarding neurons. For pruning of the insignificant neurons, sign variance strategy as described in subsection 2.4.2 is used and neurons whose variance exceeds a predefined threshold are removed from the network. The strategy is realized by sign variance tracker that observes the β values and calculates and manages the exponential moving average and variance. The exponentially weighted moving variance and standard deviation are calculated with the help of an incremental online algorithm by Finch [Fin09]:

$$\begin{aligned} \delta_i &= x_i - EMA_{i-1} \\ incr &= \alpha \cdot \delta_i \\ EMA_i &= EMA_{i-1} + incr \\ EMAVar_i &= (1 - \alpha) \cdot (EMAVar_{i-1} + \delta_i \cdot incr) \end{aligned} \tag{4.1}$$

, where α is the sign variance momentum and $x_i \in [-1, 1]$ is a new observed sign of a β_i.

Variational Dropout

Variational dropout is implemented as an adaptive bayesian dropout scheme that offers to learn specific dropout rates for each layer, as described in subsection 2.4.3.

Assuming a variational dropout layer is applied to a layer with output size $k \times l$, the implemented dropout layer manages the two parameter vectors $\log(\theta)$ and $\log(\sigma)$ of size $l \times 1$. The $\log(\theta)$ parameter are initialized to 0 and the $\log(\sigma)$ are initialized to reflect a specified initial droprate. Logarithmic scaling of the σ and θ parameters is used in the implementation. This decision is founded int he observation that the log scaling guarantees that $\sigma > 0$ and $\theta > 0$, which enhances the numerical stability of calculating the α values, and a logarithmic scale is used anyway for the calculation of the D_{KL} (see Equation 2.20). The forward pass of the dropout layer is performed according to Equation 2.13 and Equation 2.16. Thus, the parameters $\log(\sigma)$ and and $\log(\theta)$ are designated as the variational parameters of this implementation, which are optimized by the dropout layers during training. $\log(\sigma)$ through $\exp(2 \cdot \log(\sigma)) = \sigma^2 = \alpha \cdot \theta^2$ encodes the variance, while $\log(\theta)$ describes the mean value of the applied multiplicative noise

distribution. From these parameters, the multiplicative noise for each dropout layer can then be calculated, as well as $\alpha = \exp(\log(\sigma) - 2 \cdot \log(\theta)) = \sigma^2/\theta^2$ and droprate $p = \frac{\alpha}{1+\alpha}$, which are in turn used for pruning of the network.

During training, separate variational noise for each dropout layer and input are generated, with the aim to reduce gradient variance as much as possible by sampling and normalizing over sufficiently large batches. The implementation in this work omits simplification and optimization techniques such as the local reparametrization trick [KSW15] for a more straightforward implementation.

The scaling λ_D of the D_{KL} are initialized close to 0 and increase during training to make sure that the D_{KL} gets weighted more over time. In this way, the network can train first the reconstruction accuracy and then the sparsity of the neurons when the network already has a good idea of the input data. Together with the finetuning strategy, this procedure results in a higher accuracy to compression rate ratio than brutally pruning all neurons from the beginning.

As discussed in subsection 2.4.3, the log likelyhood, and thereby the training procedure can be influenced by the *psigma* parameter. Two different ways to obtain this parameter are analyzed: First, by treating it as hyperparameter and fixing it for the whole network and training (further referred to as static variational dropout), or second, by using a different network to dynamically predict *psigma* for each spatial position (hereafter referred to as dynamic variational dropout).

Finally, it is important to note that the compressive capabilities of variational dropout rely on the ability to converge the dropout rates of individual neurons to either 0 or 1. Neurons with dropout rates of 1 then transmit largely random feature information and can be pruned without affecting the predictive accuracy of the network. In order to further enhance the compression quality of variational dropout, minimizing the variance of dropout rates in the layers is thus desirable. To regulate the randomness and disorder in the dropout distributions of variational dropout layers, an additional entropy loss is introduced to the network optimization process. Given a dropout layer with n input neurons and corresponding drop rates p_i, the Entropy H is defined as

$$H = \sum_i p_i \cdot \log(p_i) + (1 - p_i) \cdot \log(1 - p_i) \tag{4.2}$$

where H is to be minimized for each drop layer.

4.2. fv-SRN

In the second approach, fully connected layers are combined with a latent feature grid to implicitly represent the input data, as proposed in fv-SRN by Weiss et al. [WHW22] (see subsection 2.3.2).

In this case the latent feature grid holds feature vectors and thus high level information about the encoded data at vertices of a grid. Since most parameters are stored in a grid instead of a fully connected network, this approach enables faster training and inference of the implicit representation than the previous Smallify-based approach. The goal of this thesis is to use pruning algorithms to prune the grid parameters so that the network automatically learns the best feature grid size for an optimal tradeoff between quality and compression, regardless of the user's knowledge.

With the goal of increasing the parameter sparsity, compactness and efficiency of the grid, fv-SRN is extended and feature grid is transformed and encoded in the frequency domain. By applying frequency transformations, a majority of the energy from the original signal is concentrated into a small number of coefficients. This allows pruning methods to sparsify large parts of the frequency coefficients without significantly impacting reconstruction quality, thereby resulting in more compact representations than when handling spatial grid coefficients. In this thesis the discrete wavelet decomposition (see subsection 2.2.3) is used to encode the feature grid. The efficiency of wavelet representation in capturing both local and global feature information has been shown in several high-performance standard codecs (e.g. JPEG2000 [TM02] or Rho et al. [Rho+22]), and therefore provides a good encoding format when pruning methods are to be used to maximize the sparsification of grid features.

To create an fv-SRN instance, the user specifies the encoding data set, an initial grid size and feature dimension, as well as the size and complexity of the fully connected predictor network. Other hyperparameters, such as the choice of embedding function, the type of wavelet decomposition or the learning rate can also be adjusted. Given a dataset X generating samples of size d, an embedding function of output size e and a latent feature grid with l features at each vertex, the predicting network consists of an input layer of size $c \cdot k$ with $c = d + e + l$, an output layer with size $k \cdot 1$ and hidden layers with size $k \cdot k$. For the neural network training, the user specifies a set amount of passes over the network and the spatial domain of the original dataset is sampled at random positions p to be fed into the network. When accessing the latent feature space, the feature grid is first reconstructed into a spatial representation from the stored wavelet coefficients and interpolated at p_i. Network parameters are again optimized by ADAM [KB14], which the user can influence by specifying or adjusting a learning rate strategy.

4.2.1. Pruning Methods

Contrary to the pruning techniques described in the Neurcomp-based network, the objective for fv-SRN is to introduce pruning layers to the latent feature grid instead of the network layers. This is motivated by the fact that a significant proportion of the

network parameters are situated in the grid, and pruning is intended to optimize the compressiveness of fv-SRN by limiting the grid size while considering the prediction accuracy of the network.

During wavelet decomposition of the feature grid, the wavelet coefficients are split into multiple grids for each decomposition level, which are each observed by a pruning layer. In this way, the pruning algorithm learns which feature vectors at which positions are significant to the reconstruction effort and omits the insignificant ones. Similar to the Neurcomp method, the pruning layers are trained simultaneously with the rest of the network. The pruning layers are also multiplied with the corresponding wavelet coefficients and then removed from the network after training. Additionally a binary pruning mask is maintained for each network. The pruning mask encodes the original structure of the grid as well as pruning information and is used for inference and reconstructing the original feature grid from the pruned state.

Again finetuning is used to further increase the reconstruction accuracy of the network, and the maximum number of training runs is split similar to Neurcomp. Two thirds of the training capacity is used for training with pruning layers, while the remaining one third is used after pruning to allow the network to readjust to its pruned state.

Trainable Binary Mask

The simple binary pruning mask is implemented as described in subsection 2.4.1 and a pruning layer is assigned to each level of the wavelet decomposition of the feature grid.

In this implementation of the method, each binary pruning layer is represented as an element-wise binary mask that activates or deactivates feature elements of a previous fully connected network layer. As described before, a stop-gradient operator is used to ensure that no gradient is backpropagated along the pruning layer and incoming gradients are passed directly to the next layer during network component optimization. During training, the mask values M are cast to binarized masks and multiplied with their corresponding input W from the previous layer. The resulting masked features W_m are then used for further interpretation in the network:

$$\begin{aligned} mask &= \sigma(M) \\ W_m &= sg(W \cdot (mask >= thresh) - W \cdot mask) + (W \cdot mask) \end{aligned} \tag{4.3}$$

where sg denotes the stop gradient operator, σ represents the sigmoid function and $thresh$ a defined threshold value for the pruning function.

Given a level of the wavelet decomposition with a grid of the dimensionality (s, c^3), consisting of the amount of subbands s and the dimensionality of the grid at level c^3, each pruning layer administers a mask of size (s, c^3). The pruning layers are accessed

when decoding the spatial representation of the grid from the wavelet coefficients and are represented as element-wise binary masks, that activate or deactivate feature elements of the latent grid.

Smallify

The Smallify pruning layers are implemented similarly to the Neurcomp architecture (see subsubsection 4.1.1), except that Smallify is used to prune the latent feature grid rather than the layers of the network. In this manner, a pruning layer is assigned to each level of the wavelet coefficients, which manages β values for each feature coefficient of the feature grid.

Variational Dropout

The variational dropout layers are again implemented similarly as for the Neurcomp architecture (see subsubsection 4.1.1). The same hyperparameters are used and only the placement of the dropout layers is changed to work on the latent feature grid instead of the fully connected network.

5. Experiments

In this chapter the quality of the simple binary masking pruning algorithm (see subsection 2.4.1), Smallify (see subsection 2.4.2) and variational dropout (see subsection 2.4.3) applied to Neurcomp (see subsection 2.3.1) and fv-SRN (see subsection 2.3.2) are assessed. In addition, the internal learning process of these pruning methods is studied and different approaches to optimize the network architectures for maximum quality-compression gain in conjuncture with the pruning algorithms are evaluated.

5.1. Experiment Setup

The goal of this thesis is to investigate the influence of the various pruning algorithms on the compression ratio and prediction quality of the Neurcomp and fv-SRN neural networks. Specifically, the sparsifying properties of the pruning algorithms on both networks are investigated. Therefore, to allow a better comparison between the two network architectures, only the unquantized states between Neurcomp and fv-SRN networks are compared, even though both approaches offer the possibility to quantize the network architecture after training to achieve a better quality-compression ratio. To make sure that the findings in this thesis are also applicable to the quantized states of the networks, Figure A.1 compares pruning results for both networks prior and post applied quantization. The figure indicates that the relations between baseline and the pruned versions of the network do not change after quantization. This validates examining the pruning effects on the networks in their unquantized states.

Experiments are performed on two 3D scalar data sets: a small turbulence data set of size $(150, 150, 150)$ and a larger pressure magnetic field (mhd_p) of size $(255, 255, 255)$ from a magneto-hydrodynamicisotropic turbulence simulation of the Johns Hopkins Turbulence Database [Li+08]. All compression results of the neural networks can be written out as vti files and visualized with appropriate volume rendering software, such as paraview [Ahr+05]. Renderings of the mhd_p and turbulence data sets can be found in Figure A.2.

To obtain the best possible deterministic results, first the number of iterations required for convergence for each combination of data set and network is determined, and then the number of iterations is set constant for each experiment. The number of iterations is therefore set to 80 - 100 for Neurcomp, while the fv-SRN converges much faster

with 40 - 60 iterations. In a similar manner, hyperparameter search is utilized to find good constant values for batch- and sample size throughout all experiments. For the fv-SRN, an empirical decision is made to use a Fourier feature embedding and the Haar wavelet filter for all experiments. This is done to restrict the dimensionality of the the search space for NAS and to obtain more consistent results. However, the majority of hyperparameters, such as the learning rate, weighting terms for different losses, parameters for the various pruning methods, and the overall architecture of the network, such as the number and size of the network layers, are dynamically determined by the AX NAS algorithm and elaborated on in more detail for each experiment.

For all experiments the Peak signal-to-noise ratio (PSNR) is used as a quality measure for the accuracy of the network predictions and the compression ratio of the network to the original data is utilized as a measure for the compressive capabilities of the network. The PSNR for each network prediction is calculated from the computed mean square error (MSE) between the original scene data and the entire implicitly reconstructed scene. The compression ratio is derived from the difference between the amount of parameters used in the original data and the implicit model.

5.2. Classical compression baseline

To assess the compressive capabilities of the base Neurcomp and fv-SRN networks without pruning algorithms, both algorithms are compared against TTRHESH (see subsection 2.2.1) as the selected state-of-the-art compression method without deep learning. The use of TTHRESH as a baseline for classical, non-learning compression algorithms is motivated by its high reconstruction accuracy and ability to compress smoothly at dynamic target compression ratios. TTHRESH is able to surpass other classical compression algorithms in regard to obtained reconstruction quality and is used as a baseline in other recent studies [Lu+21], [WHW22], [HWW22].

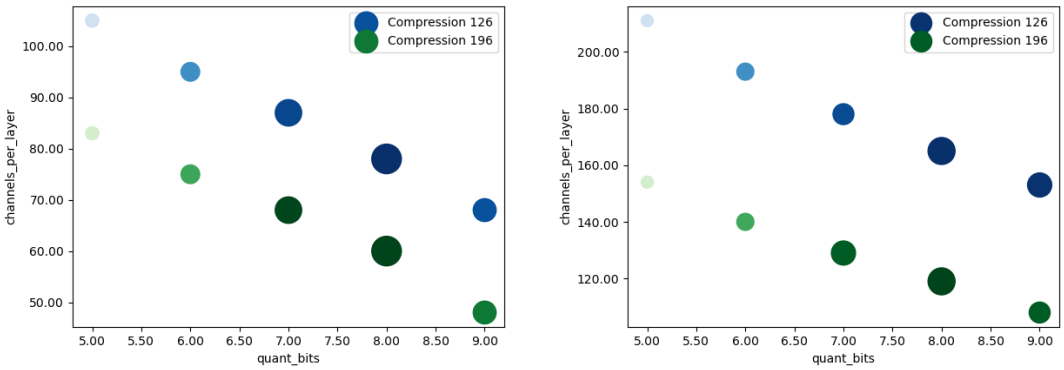
First, the effect of quantization on the reconstruction quality and compression ratio on Neurcomp is investigated. To this end, a comparison is made between the obtained PSNR value when setting the compression rate and weighting the size of the network against the permitted quantization accuracy. To find the best tradeoff between quantization compression and accuracy, Neurcomp is trained on both the turbulence volume and the mhd_p data set. For each data set, two compression rates are selected and the correct size of the network layers is calculated, to achieve the corresponding compression rate after quantization with different quantization accuracy. Figure A.1a visualizes the findings for this experiment by depicting the allowed quantization accuracy in form of the permitted storage bits for each network parameter on the x-axis, and the size of each network layer on the y-axis. Each dot in the figure represents a compression of

the original data set, where the opacity and size of the dot encodes the achieved PSNR. Interestingly, both data sets show that the overall compression quality increases with the allowed quantization precision, up to 8 bits per parameter. These findings suggests that quantizing each network weight with 8 bits yields the best results, and thus these settings are used for the following baseline comparison. For fv-SRN the specifications from the original paper are adopted. As such the feature grid values are quantized with 8 bits and the network parameters with 16 bits.

For comparing the Neurcomp and fv-SRN network with TTHRESH, the AX NAS algorithm is employed to find the pareto frontier of both network architectures. The batch size, sampling size, quantization accuracy for Neurcomp and fv-SRN, and choice of wavelet transform are fixed, as is the choice of embedding function for fv-SRN. Then the NAS algorithm is executed with each 80 search iterations and decides on the learning rate, learning rate decay, number of network layers, size of network layers, and in the case of fv-SRN the size of the feature grid for varying compression ratios. Figure 5.2 shows the results of this hyperparameter search by depicting the achieved compression ratio on the x-axis, the PSNR as a quality measure on the y-axis, and the different compression approaches as colored linear graphs. While the TTHRESH baseline is able to outperform both neural networks in their unquantized representations, Neurcomp and fv-SRN achieve superior results than TTHRESH when coupled with quantization of the network parameters. Since compressive neural networks can outperform TTHRESH as a state-of-the-art method for compressing scientific data, these results indicate that further research and development in refining learning based compression approaches is justified.

5.3. Pruning on Neurcomp

In this section, Smallify and variational dropout are explored as methods to improve the compressive capabilities of Neurcomp. The key to this goal is to find the best balance between using the pruning algorithms to remove neurons from the network, while retaining most of the compression quality. To this end NAS is used to optimize the choices for the network architecture (amount and size of hidden layers, learning rates) and to get a good grasp of the influence of the hyperparameter of the pruning methods. Figure 5.3 illustrates the results of this broad multi-objective optimization. The figure plots and compares the pareto frontiers along PSNR and compression ratio of the variational dropout and Smallify approach with the baseline of the unquantized, unpruned Neurcomp architecture. For the variational dropout, the approach of statically defining the parameter *psigma* as a hyperparameter selected by the NAS (static variational dropout) is compared with keeping *psigma* as a dynamic prediction of a



(a) turbulence volume data set

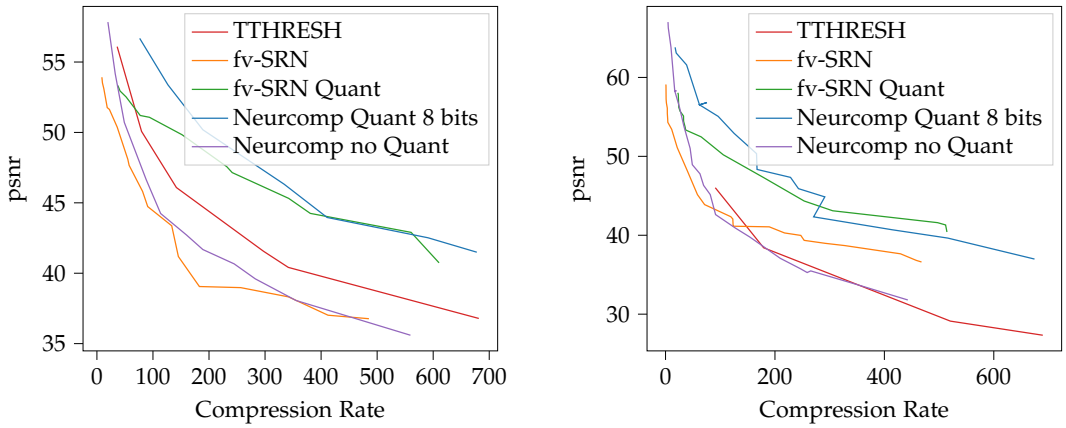
(b) mhd_p data set

Figure 5.1.: Comparing different combinations of quantization accuracy and network complexity in regard to their reached PSNR (encoded in the albedo and size of the dots). The final compression rate after network training and compression is fixed, and the network complexity is altered according to the permitted quantization precision. For both the mhd_p and the turbulence data set, the best results are obtained with a quantization precision of 8 bits.

second, smaller network (dynamic variational dropout). The experiment is run with the small turbulence volume data set and the larger mhd_p data set. Additionally, multiple experiments are conducted for each pruning and baseline method for the turbulence volume data set. This is done to get a better understanding of the variance between the different runs and enable more sophisticated comparisons.

The compressive abilities of Neurcomp are not optimized from pruning on the smaller turbulence volume data set (see Figure 5.3b). All pruning algorithms behave relatively similar and are not able to surpass the unpruned baseline in regard to quality-compression ratio. The considerable variance observed in the experiment runs can be attributed to the vast high-dimensional search space of the network and pruning hyperparameters that need to be evaluated by the NAS. Although the variance is not large enough that a comparison between the pruning methods and the unpruned baseline is not feasible, it must still be taken into account when evaluating the benefits of the pruning methods.

Evaluation of the pruning methods on the mhd_p data set (see Figure 5.3a) shows better results for the potential of pruning methods. On the larger data set all pruning approaches are capable of outperforming the baseline, although only the static vari-



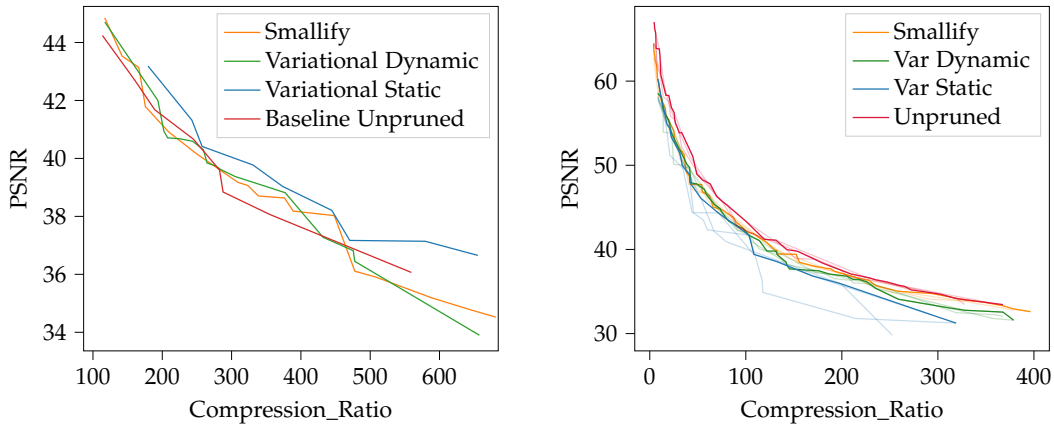
(a) Baseline for mhd_p data set

(b) Baseline for turbulence data set

Figure 5.2.: Comparison of the TTHRESH, fv-SRN and Neurcomp (once without quantization, once with quantization) in regards to their obtaines PSNR and compression ratio on the mhd_p and turbulence data sets.

ational approach consistently surpasses the baseline by between 1 to 2 PSNR points. Both Smallify and dynamic variational dropout algorithms demonstrate unstable performance and can only marginally outperform the baseline by 0.5 to 1 PSNR points in some cases. Due to the variance observed in the turbulence volume experiment, it is inconclusive whether these pruning algorithms offer significant improvements over the unpruned baseline.

A second experiment is initiated with the goal to generate a better comparison of the pruning techniques. In order to better control the variance of the NAS algorithm, the hyperparameter searchspace is scaled down and the Neurcomp network architecture is set constant at a small compression ratio. Only the pruning hyperparameters are varied to obtain increasing compression ratios on the mhd_p data set. Figure 5.4 displays the compression results acquired by static variational dropout (Figure 5.4a) and Smallify methods (Figure 5.4b) for comparison. Smallify is tested on two starting architectures, one starting with a compression ratio of 100 and the other with 200. While the pruning for the smaller network with the larger compression ratio performs a little bit better overall, both struggle to surpass the unpruned network and follow the baseline when tasked with pruning large chunks of the original network for larger compression ratios. Interestingly, Smallify performs best when the original network is only slightly pruned, increasing the prediction quality and compression ratio of the base network. Variational dropout, on the other hand, underperforms for small compression ratios when tasked to prune tiny parts of the network, but is able to outshine the baseline and Smallify for



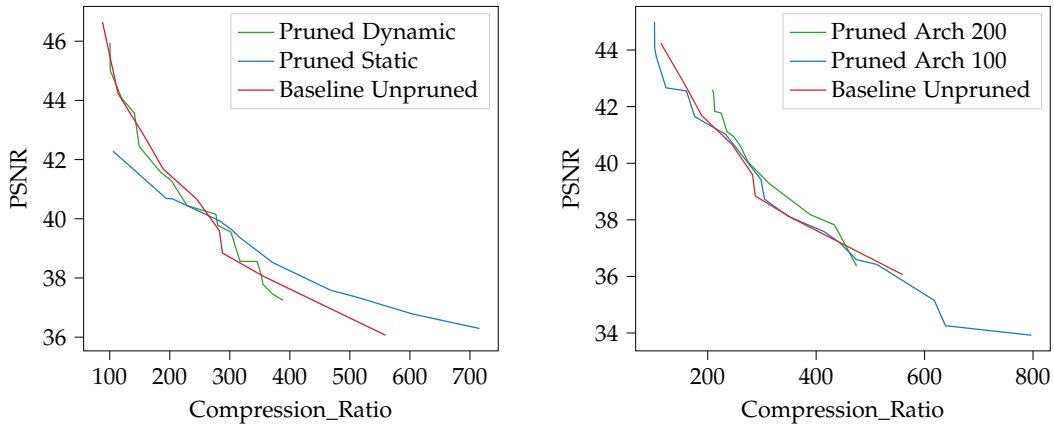
(a) mhd_p (b) turbulence volume

Figure 5.3.: Comparison of prediction accuracy (PSNR) and compression ratio of variational dropout along with Smallify against the unpruned Neurcomp baseline on the mhd_p and turbulence volume data set. All plots are generated by employing NAS to search for optimal network and pruning hyperparameters. Each experiment was repeated 3 times for the turbulence volume data set to generate a better understanding of the variance in the data (faded plots).

larger compression ratios and larger pruning amounts.

Hyperparameter Analysis

In order to understand the pruning behaviour of the different pruning methods, the influence of various pruning hyperparameters on the compression process are investigated. To this end, the Neurcomp architecture is set constant and NAS is used to find the pareto frontier of the pruning algorithms in regard to PSNR and compression ratio. Trends of the hyperparameters along the pareto frontier are then analyzed. Figure 5.5 visualizes the results for this experiment by mapping various values of the pruning hyperparameters on the x-axis against the final compression ratio on the y-axis. Not all hyperparameters appear to have a meaningful effect on the overall realized compression ratio, and many parameters seem to be either set to singular values for multiple compression ratios or change their values seemingly at random. The experiment is complicated by the considerable variance present in the NAS data, due to the high dimensionality of the hyperparameter search space and the limited sample size of the NAS algorithm. These circumstances make it a challenging task to



(a) Variational Dropout

(b) Smallify

Figure 5.4.: Comparing the obtained PSNR and compression ratio of Neurcomp when enhanced with the Smallify or variational dropout algorithms. The network hyperparameters are fixed for each plot and NAS is used to find the optimal pruning hyperparameters with different amounts of pruning and thus different compression ratios.

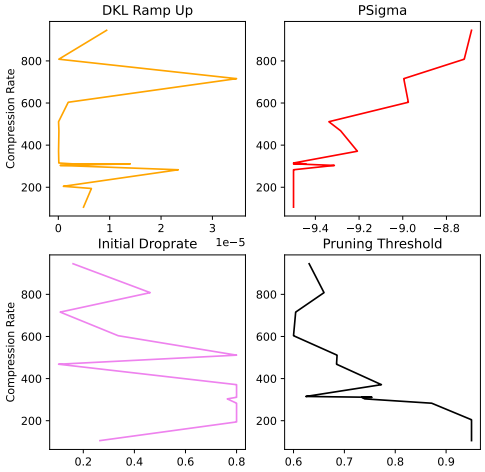
differentiate valid trends along the pareto frontier from noise values.

To this end, the most important hyperparameters for each pruning method can be narrowed down by following and extracting the gradient of each plotted hyperparameter function, where the parameters with the strongest gradient have the largest influence on the pruning behaviour of the algorithm. It is found that the static variational dropout is largely influenced by the choice of *psigma*, which controls the sensitivity of the log likelyhood loss and the pruning theshold choice. According to the depicted function in Figure 5.5a, a larger *psigma* corresponds to more aggressive pruning and a higher compression ratio. In contrast, a larger pruning threshold corresponds to a decreased final compression ratio.

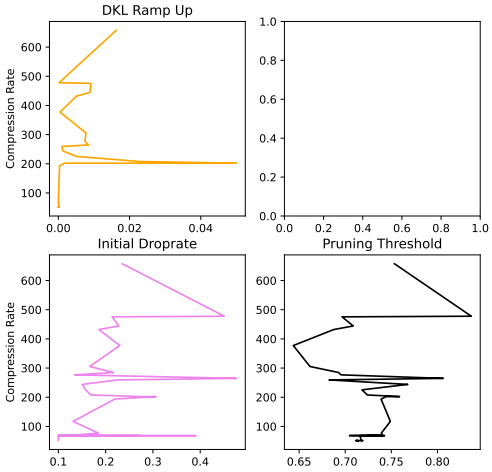
Dynamic variational dropout, on the other hand, uses an intern neural network to predict this hyperparameter and depends on the rate at which the D_{KL} gains importance for the overall optimization (see Figure 5.5b). Here, a faster ramp up of the D_{KL} influences a larger final compression.

Finally, Smallify appears to be dependent on both the weighting term for pruning loss and weight loss (see Figure 5.5c). Increased weighting of the beta and weight loss correspond to increased final compression.

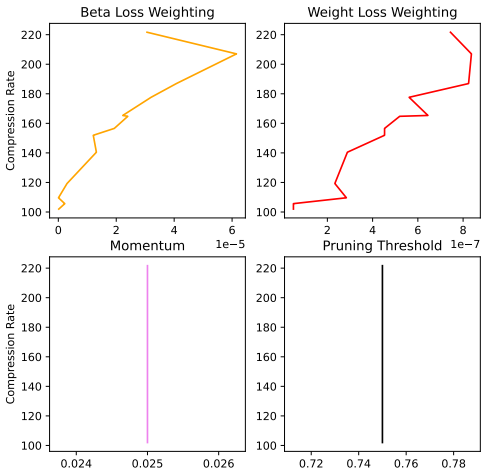
From these findings simple rules can be generated that allow the user to select the important hyperparameters corresponding to a chosen compression ratio or PSNR



(a) Static Var



(b) Dynamic Var



(c) Smallify

Figure 5.5.: Comparison of hyperparameters of the corresponding pareto frontiers for static and dynamic variational dropout, as well as Smallify. Each hyperparameter is increased along the x-axis and mapped against the resulting compression ratio on the y-axis.

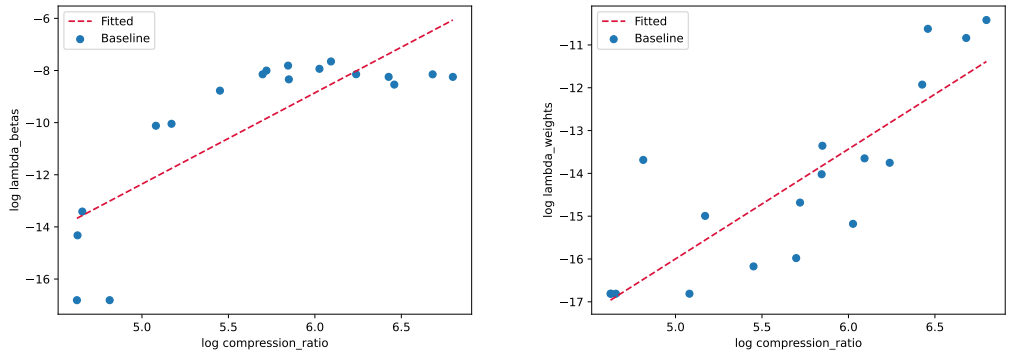
value for each pruning algorithm. The results of this task are depcited in Figure 5.6. The problem is tackled with linear regression and a curve fitting algorithm is used to generate simple functions that best fit the observed data points from Figure 5.5. For the Smallify and variational dropout methods, all examined hyperparameters are positive. Thus a log scaling of the parameters and compression ratio is applied to minimize the influence of outliers and combat the noise in the data. Figure 5.6a visualizes the results of this experiment for Smallify. According to the hyperparameter analysis, two curves are fitted for the weighting of the beta and the weight loss. Both parameters can be approximated by underlying linear functions, although the weighting term for weight loss has less noise and can be better approximated.

Figure 5.6b depicts the fitted curves for the static variational dropout according to *psigma* and the pruning threshold. Both hyperparameters fit well to the approximated function, especially the *psigma*, indicating a stable relationship between the hyperparameter and the pruning behaviour.

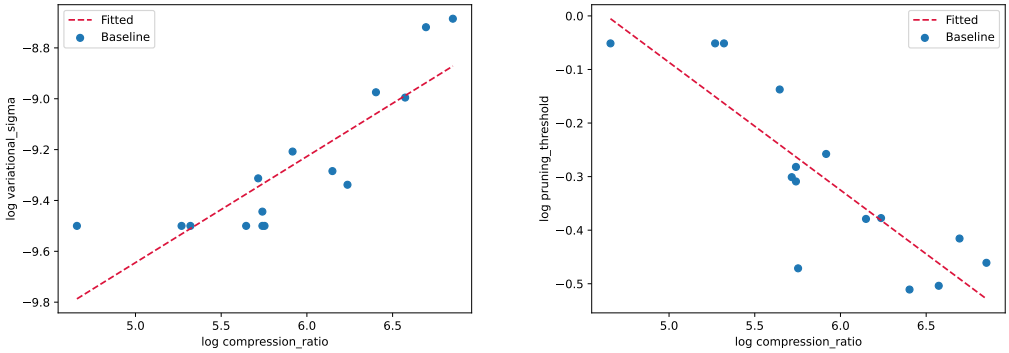
In Figure 5.6c the dynamic variational dropout is approximated by fitting a linear function on the ramp up value of the D_{KL}. While Smallify and the static variational dropout were rather easy to fit to an approximate linear function, the dynamic variational dropout baseline seems to contain many outliers and is worse to fit than the previous methods.

The quality of the predictor models is investigated by comparing the predicted theoretical compression results against real achieved compression ratios. To this end the regressed functions from Figure 5.6 are used to generate choices for pruning hyperparameters that are used with Smallify, dynamic and static variational dropout to achieve experimental runs with increasing compression ratios. Different Neurcomp models are then trained with the varying hyperparameters according to the predictor model, while all other hyperparameters are fixed between runs. Figure 5.7 visualizes this experiment by comparing the predictor functions against the real achieved compression ratios. While the results of both static variational dropout (Figure 5.7a) and Smallify (Figure 5.7c) are close to their approximated counterparts, the behaviour of dynamic variational dropout (Figure 5.7b) is poorly estimated by the fitted curve in comparison. These observations match with the estimated quality of the simple prediction models. Static variational dropout (Figure 5.6b) and Smallify (Figure 5.6a) contain relatively low variance in the data that was used to generate the predictor functions, and therefore the correlations of the hyperparameters can easily be captured and estimated by simple linear functions. Dynamic variational dropout (Figure 5.6c) on the other hand contains more noisy data, and the connections between the hyperparameters and the final compression result are hard to discover and capture by a simple linear approximation.

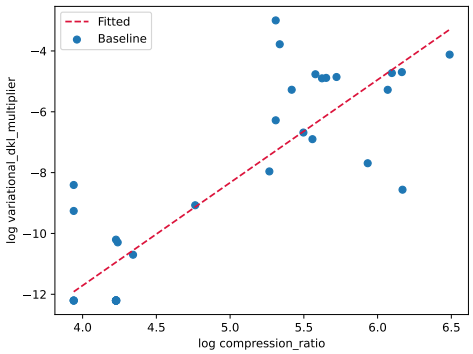
Finally, correlations between the final compression ratio of the network after pruning and the complexity of the initial network are investigated. With this insight, the



(a) Smallify

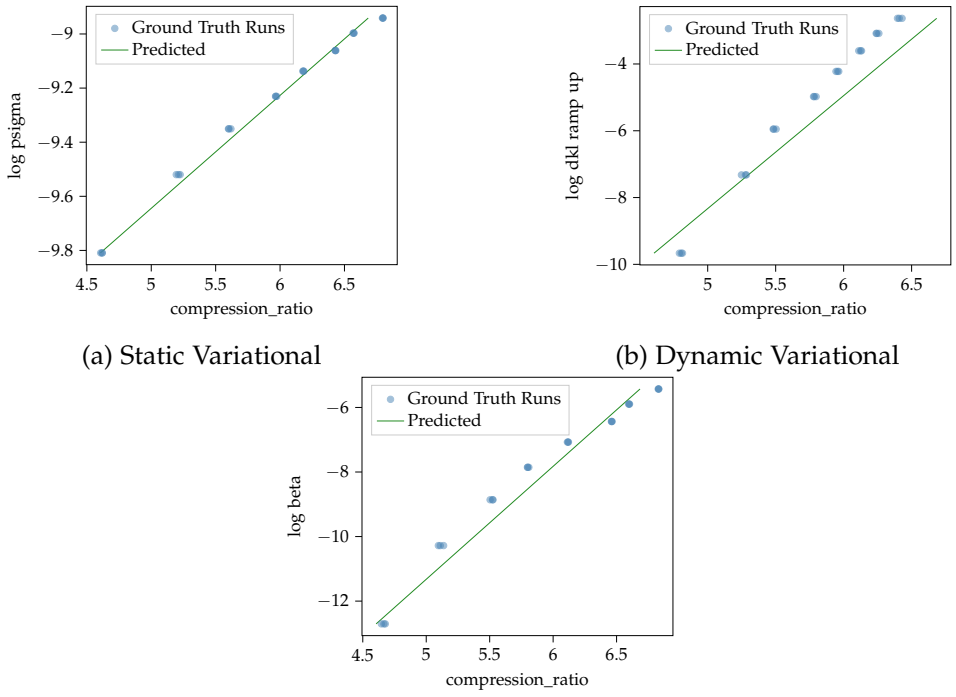


(b) Static Var



(c) Dynamic Var

Figure 5.6.: Regression of simple functions that calculate choices of hyperparameters to arrive at a given compression ratio for the Smallify and variational dropout methods. All models analyze the pareto frontier of compression runs on the mhd_p data set with Neurcomp and are fitted in log space.



(a) Static Variational

(b) Dynamic Variational

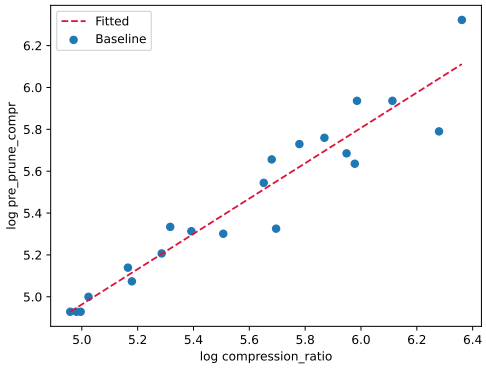
(c) Smallify

Figure 5.7.: Control quality of regressed linear models for Neurcomp. The theoretical predicted results of each linear model are compared with real compression outcomes.

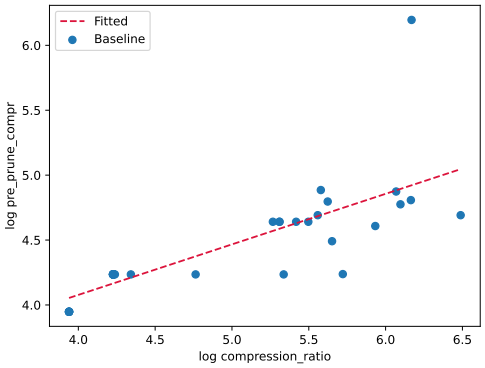
initial size of Neurcomp for each pruning method can be derived to achieve a specific compression ratio with maximum reconstruction quality. This experiment is depicted in Figure 5.8, where along the pareto frontier from Figure 5.3 the final compression on the x-axis and the initial network compression ratio on the y-axis are compared. On the basis of these data simple linear functions are again regressed. Apart from allowing users to choose the optimal starting network complexity for a given final compression rate, this experiment gives further insight in the pruning behaviour of the pruning methods for varying network complexities.

From the data of the static and dynamic variational dropout (Figure 5.8a, Figure 5.8b), functions can be regressed that initially map final compression ratios to starting compression ratios of similar size. This indicated that these dropout methods prune relatively small bits of the original network at small target compression rates. However, this changes at larger compression ratios of 200 and above, where especially dynamic variation dropout cuts off large portions of the originally larger network. Smallify

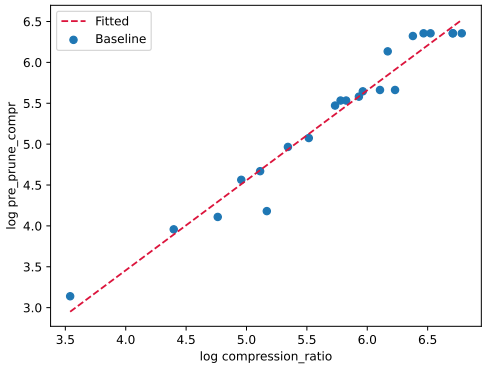
(Figure 5.8c), on the other hand, maintains a relatively stable correlation between the final and initial network complexity along the examined compression rate range.



(a) Static Variational



(b) Dynamic Variational



(c) Smallify

Figure 5.8.: Correlate the final and initial network compression ratio for variational dropout and Smallify. All data points were taken from the pareto frontier of a free hyperparameter search of Neurcomp on the mhd_p data set.

Effects of Resnet Architecture

The results of Figure 5.4 and Figure 5.3 show that pruning methods, especially variational dropout, can indeed increase the quality-compression ratio of Neurcomp. It is reasonable to assume that even more improvements can be made by further unlocking the potential of the pruning methods. A major weakness of the Neurcomp architecture

towards pruning algorithms is its reliance on residual blocks. Although these blocks bring stability to the training of deep neural networks, they also impose restrictions on the structure of the models. In particular, they require that the incoming and outgoing neurons of a residual block have the same size so that residual features can be correctly transmitted. This restriction limits the use of pruning layers to be applied only within the residual blocks, effectively halving the amount of pruning layers in the network. Removing these residual blocks risks destabilizing the training process, but allows to supervise all hidden layers of the network with pruning layers. This tradeoff is investigated by comparing the Smallify and variational dropout methods with and without the use of residual layers.

Figure 5.9 visualizes the result of this experiment for the Smallify method. The NAS algorithm is again used to generate a baseline, as well as optimal parameterization for the pruning network. Similar to the experiment results in Figure 5.4b, Smallify is able to surpass the compression baseline for very small amounts of pruning. Without the restrictions of the resnet-architecture, however, Smallify can also outperform the baseline and pruned resnet version for larger compression ratios over 200. Nonetheless, these compression gains are still quite small, with only between 1 and 1.5 PSNR points difference and are therefore not exempt from influence through NAS variance.

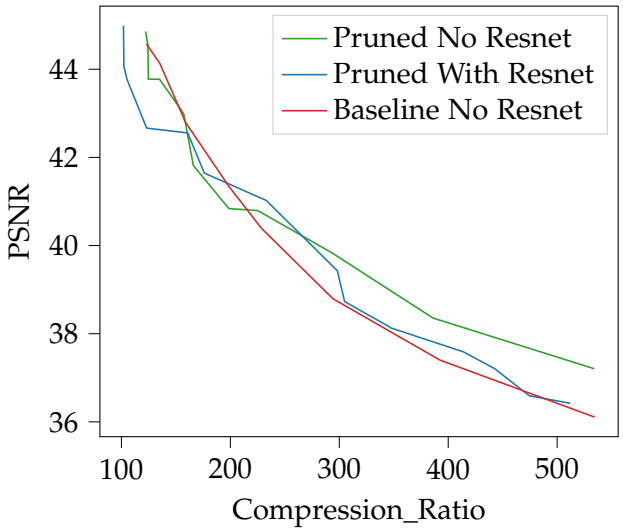
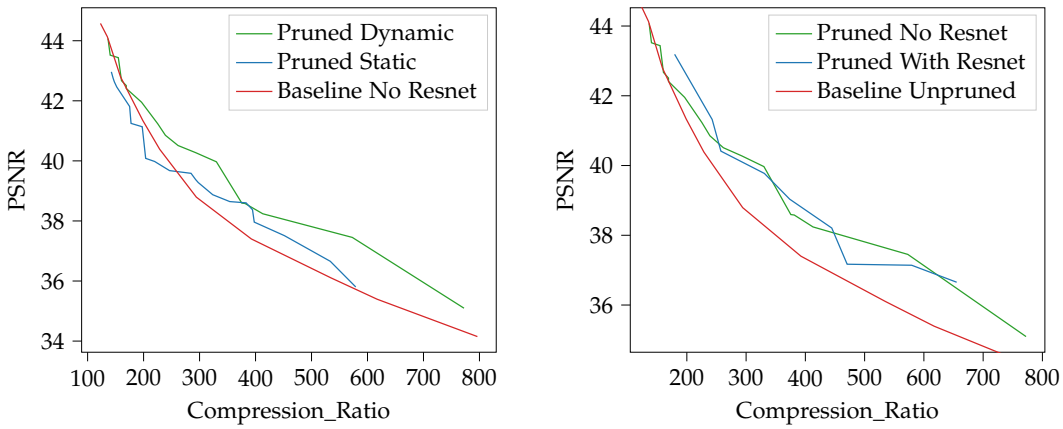


Figure 5.9.: Comparison of the quality-compression ratio when using the Smallify technique on the same Neurcomp network with and without built-in residual blocks.

Figure 5.10 depicts the results for the same experiment with the variational dropout

approach. The figure first compares dynamic and static variational dropout against the unpruned baseline (Figure 5.10a). While both methods are able to surpass the baseline, in this experiment the dynamic pareto frontier is able to exceed the static pareto frontier, contrary to the results of the previous experiment, where residual blocks are enabled (see Figure 5.3a). The dynamic variational dropout case without residual blocks is compared with the previous best case of static variational dropout with residual blocks in Figure 5.10b. This experiment shows that while the approach of omitting residual blocks benefits dynamic variational dropout, the loss in training stability outweighs this gain, as both methods are relatively equal in terms of their quality-compression ratio.



(a) Static VS Dynamic Pruning

(b) Pruning with and without Resnet

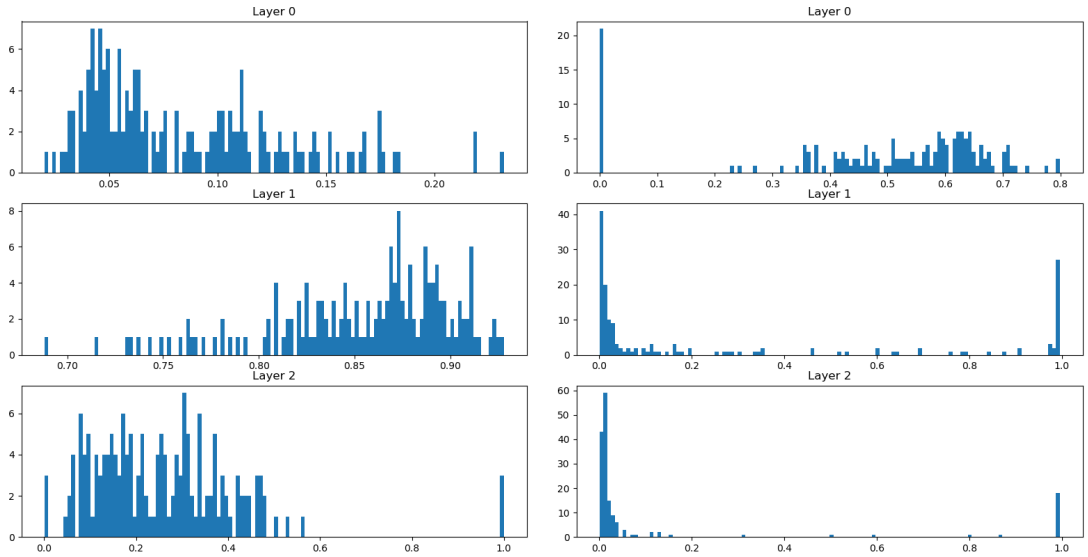
Figure 5.10.: Analysing the effects of variational dropout on Neurcomp with and without residual blocks in regard to the final reconstruction quality and compression ratio. Right: Comparison of dynamic pruning without resnet vs best results from NAS on Neurcomp with resnet (static pruning).

Variational Dropout With Entropy Loss

As motivated in subsubsection 4.1.1 this thesis aims to further enhance the compressive capabilities of variational dropout by introducing an additional entropy loss to the training process. By incorporating the entropy of each dropout layer into the final optimization, the goal is to improve the convergence of the dropout layers towards the extrema of 0 and 1, leading to more consistent pruning decisions.

The effect of the Entropy loss on the optimization process is investigated by stopping the network training before the pruning of the dropout layers and examining the

resulting droprate distributions of each dropout layer. Figure 5.11 compares the dropout distribution of each node for variational dropout compression with (Figure 5.11b) and without (Figure 5.11a) the use of the additional Entropy loss. For both runs all other network and pruning hyperparameters are kept constant and after initial NAS the weighting of the Entropy loss is empirically set to be equivalent to the weight loss. The experiment without the additional entropy shows that the majority of droprates converge to either 0 or 1, but the variance in dropout values is still high. On the other hand, drop rates for the experiment with entropy loss show better convergence to these extremes than the drop rates in the experiment without entropy loss, although they still are not perfectly distributed between 0 or 1, and especially layer 1 shows a large variance in drop rates.



(a) Without Entropy Loss

(b) With Entropy Loss

Figure 5.11.: Comparison of the distribution of dropout rates of variational dropout layers in a Neurcomp network trained with and without additional Entropy loss. Both networks are trained on the mhd_p data set until convergence with otherwise fixed hyperparameters. Droprates in the network with additional entropy loss converge more uniformly to the extrema 0 or 1 than in the network without entropy loss.

In addition, the effect of the Entropy loss on the training effectiveness is investigated by comparing the PSNR and compression ratio of variational dropout networks that incorporate the Entropy loss with those that do not. In Figure 5.12 this experiment is visualized for networks with fixed hyperparameters for varying compression ratios.

Although the Entropy loss has an impact on the droprate distribution, it falls short in significantly impacting the final compression result. While the final compression ratios and PSNR values vary slightly, these fluctuations can likely be attributed to variance and irregularities in the network training process.

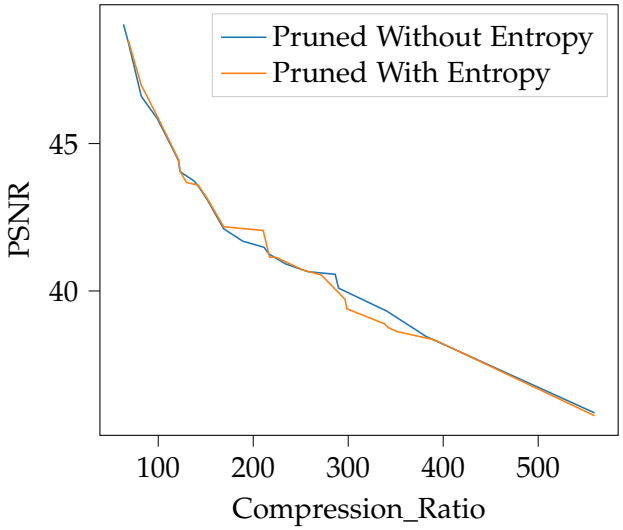


Figure 5.12.: Comparison of achieved PSNR and compression ratio when training dynamic variational droupout with and without the additional Entropy loss on the mhd_p data set.

Layerwise Thresholds on Variational Dropout

Using the different dropout distributions in Figure 5.11a, it is observable that different variational dropout layers in the Neurcomp architecture converge in different ways. Based on this observation, it is proposed to use different thresholds for each layer instead of a single global threshold used for the entire network. To this end the same unpruned variational dropout Neurcomp network as in Figure 5.11a is analysed and the effects of different layer thresholds on the PSNR and compression ratio are investigated. Figure 5.13 depicts this experiment as a parallel coordinate plot, where varying thresholds for each dropout layer are put in relation to the resulting PSNR and compression ratio for a fixed and already trained network. The goal of this experiment is to identify the thresholds that result in the highest combination of compression ratio and PSNR.

The choice of a threshold for the first layer is relatively meaningless, as samples that result in the highest PSNR and compression ratio can be drawn from the whole

threshold value range in layer one. In contrast, the choice of threshold for the second and third layer has a large impact on the resulting PSNR and compression ratio. Setting the threshold for the third layer too high (e.g., below 0.4) will result in poorer prediction quality, while insufficient pruning in the second layer (e.g., with a threshold above 0.8) will also result in worse compression rates. This is consistent with the corresponding observed droprate distributions, as layer three indicates all droprates below 0.4 to be converging towards 0, and all droprates above 0.8 to be converging towards 1 in layer 2 given sufficient time. This experiment shows that layerwise thresholds improve the quality-compression ratio of variational dropout in Neurcomp.

5.4. Pruning on fv-SRN

The goal of the experiments in this section is to improve the compressive capabilities of fv-SRN using the simple binary masking, Smallify, and variational dropout. The process is conducted similarly to Neurcomp, where NAS is used to establish a good baseline for the unpruned network and to gain insight into the impact of pruning hyperparameters on the compression process. Figure 5.14 shows the results of the multi-objective optimization by plotting the achieved compression ratio on the x-axis and the PSNR on the y-axis along the pareto frontier of each NAS run. The pruning algorithms and the unpruned baseline are compared on the smaller turbulence volume and larger mhd_p data set. To more accurately capture the existing variance in the NAS, multiple experiments are being conducted again on the turbulence volume data set.

The results for the turbulence volume data set are depicted in Figure 5.14b. The singular experiment runs are varying strongly in achieved compression ratio and PSNR, indicating a high variance in the compression results on the fv-SRN. Nevertheless, when comparing the overall strongest compression results for the unpruned baseline and each of the pruning algorithms, an advantage of the pruning algorithms over the baseline from 2-5 PSNR points can be seen at smaller compression ratios up to 60. At higher compression ratios, this advantage becomes less clear and the high variance makes it difficult to make definitive conclusions about the advantage of pruning methods over the unpruned network.

Figure 5.14a illustrates the performance of the pruning algorithms on the larger mhd_p data set. Compared to the smaller turbulence volume data set, the pruning algorithms demonstrate a superior compression gain. Smallify, the simple binary masking and dynamic variational dropout surpass the unpruned baseline by 4 to 5 PSNR points between target compression ratios of 80 to 150. For very small, or considerably larger compression ratios upwards of 250 the quality gain for all pruning algorithms

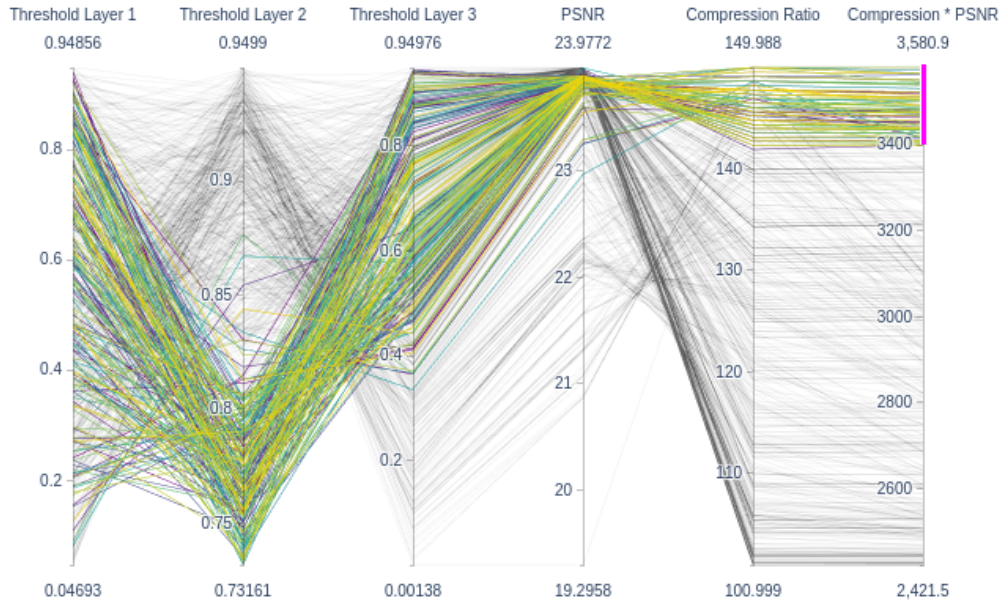
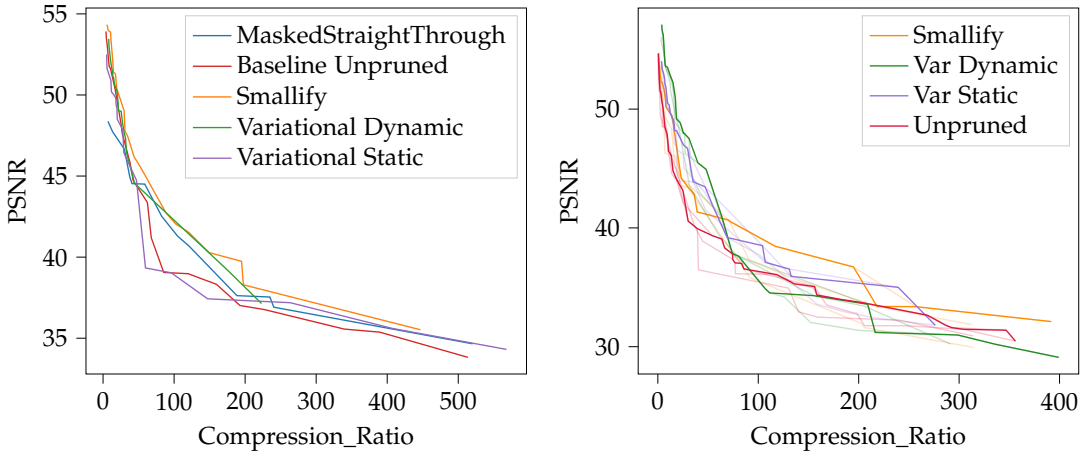


Figure 5.13.: Effect of different dropout thresholds on variational dropout. In this experiment Neurcomp is enhanced with dynamic variational dropout and trained until convergence. Random dropout thresholds for each pruning layer are then sampled uniformly and each layer is pruned accordingly. The results for the optimal combination of PSNR and compression ratio, which is attained by varying the dropout thresholds in layer 2 and layer 3, are analyzed. The varying optimal thresholds suggest that layerwise thresholds are preferable to a global threshold for all layers in terms of quality-compression ratio.

slows down. Static variational dropout outperforms the baseline for larger compression ratios over 200 by a small amount, but underperforms for small compression ratios. In contrast to the results for Neurcomp, both Smallify and dynamic variational dropout consistently yield a higher compression-quality ratio than the unpruned baseline. A comparison of the performance of the pruning algorithms in relation to each other is difficult because of the high exhibited variance from the data along the pareto frontier.

To provide a comparison of pruning methods based on the least amount of variance



(a) mhd_ p

(b) turbulence volume

Figure 5.14.: Comparison of prediction accuracy (PSNR) and compression ratio of the simple binary masking, Smallify, dynamic and static variational dropout and unpruned baseline for the fv-SRN on the mhd_p and turbulence data set. NAS is used to search simultaneously for optimal network and dropout hyperparameters. Multiple NAS runs are performed for each pruning method for the turbulence data set (faded lines).

in finding the optimal network architecture, again the network hyperparameters are fixed and only the pruning parameters are changed to stimulate more pruning and varying compression ratios. The Ax NAS algorithm is used for hyperparameter search and to find the best pareto frontier of each pruning method along varying target compression ratios. This experiment is depicted in Figure 5.15. Smallify again achieves impressive results, outperforming all other pruning techniques and the baseline for compression ratios between 150 and 350. The binary mask as well as the dynamic variational dropout approach perform similar and are also able to surpass the unpruned baseline. Static variational dropout does not perform as well at smaller compression ratios up to 300, but improves at larger compression ratios. While the binary masking, Smallify, and dynamic variational dropout converge to similar quality-compression ratios at compression rates above 350, static variational dropout is able to outperform all other methods at these rates.

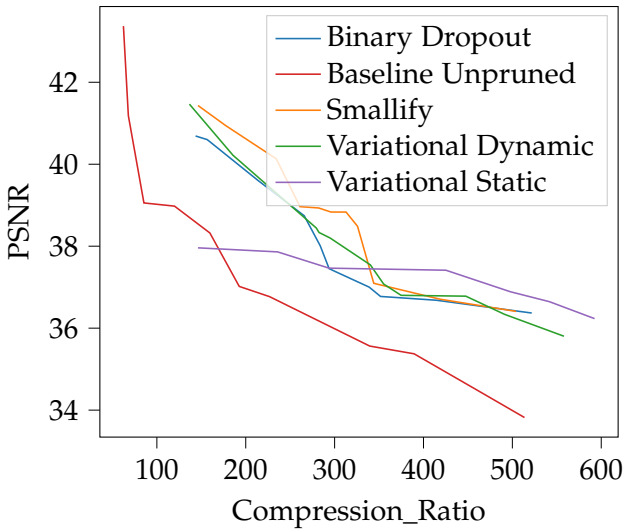


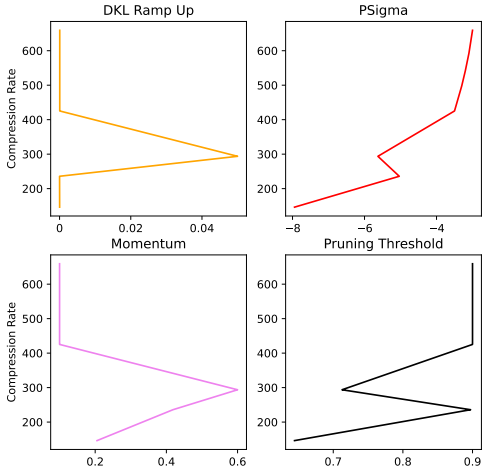
Figure 5.15.: Comparison of achieved PSNR and compression ratio when using the simple binary masking, Smallify, dynamic or static variational dropout with fv-SRN on the mhd_p data set. The network hyperparameters are fixed for all pruning methods and NAS is used to search for different optimal pruning hyperparameters to get varying compression ratios.

Hyperparameter Analysis

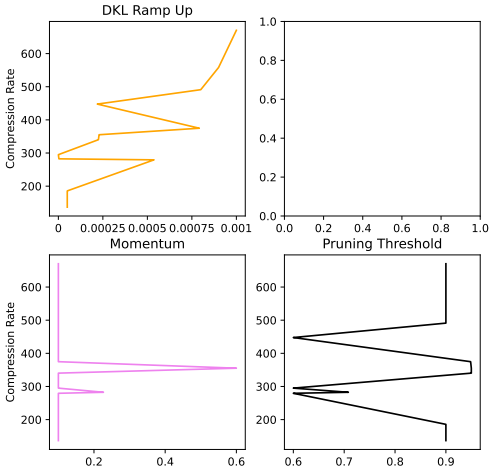
The influence of pruning hyperparameters on the pruning process is again investigated by analyzing trends of the hyperparameters of each pruning technique. The results are presented in Figure 5.16, which compares the impact of pruning hyperparameters on the achieved compression ratio after training for each pruning method along the pareto frontiers acquired in Figure 5.15. It can be observed that the pruning algorithms in fv-SRN display similar behavior to Neurcomp, as discussed in subsubsection 5.3.

Static variational dropout (see Figure 5.16a) again exhibits a strong gradient in the *psigma* parameter, which controls the extent of pruning with increasing value. Dynamic variational dropout (see Figure 5.16b) behaves more unstable. Most hyperparameters seem to switch values randomly without discernible coherence. The strongest gradient can again be seen in the ramp up term for the D_{KL}, although this plot also contains many outliers. Smallify (see Figure 5.16c) and the simple binary masking (see Figure 5.16d) are both mainly affected by the loss weighting factors for the beta-loss and weight regularization.

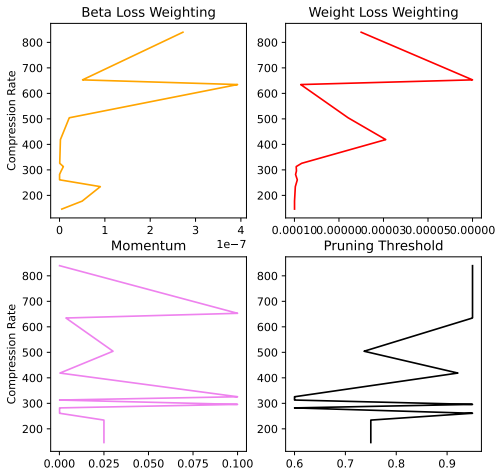
Following a similar approach to Neurcomp (see subsubsection 5.3), the findings from the hyperparameter analysis are used to derive simple function approximations for



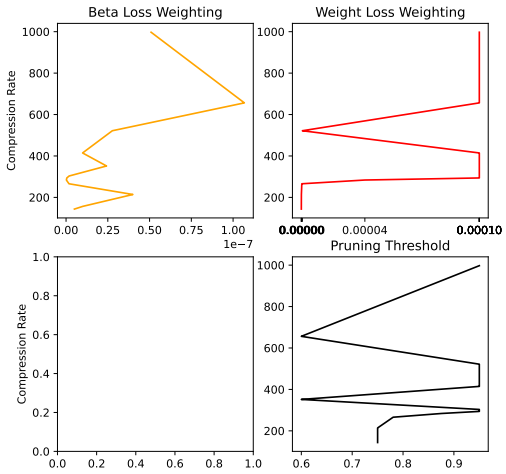
(a) Static Var



(b) Dynamic Var



(c) Smallify



(d) Binary Masking

Figure 5.16.: Comparing hyperparameters of the corresponding pareto frontiers for the static and dynamic variational dropout, as well as Smallify. Each hyperparameter is increased along the x-axis and mapped against the resulting compression ratio on the y-axis.

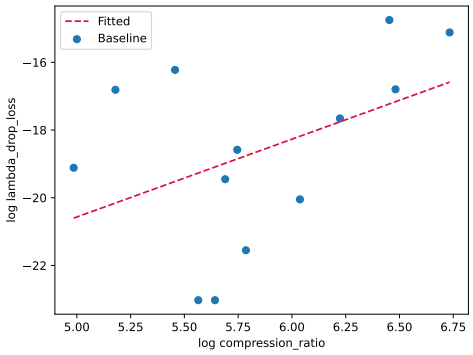
selecting a hyperparameter based on a desired compression ratio. The results of this regression analysis are presented in Figure 5.17. The considerable variance depicted by the hyperparameter search on the fv-SRN results in a high amount of noise in the data. To mitigate the impact of outliers on the fitting process, a logarithmic scaling is applied for both the compression rate and supplementary parameters, as they are non-negative.

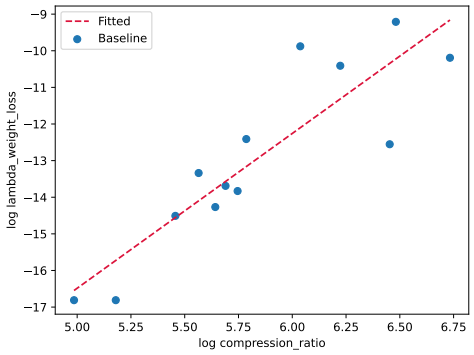
Due to the high stability of the static variational dropout pareto frontier as a function of the *psigma* parameter, the influence of this hyperparameter can be approximated relatively well (see Figure 5.17c) and a simple function can be regressed that captures the underlying trend of an increasing *psigma* corresponding to an increasing compression ratio. The dynamic variational dropout, on the other hand, contains a larger parameter variance in the data and cannot be approximated as well (see Figure 5.17b). Smallify (see Figure 5.17a) and the simple binary masking (see Figure 5.17b) both exhibit similar data trends and behave similar to the curve fitting task. Both pruning algorithms have a higher variance in the beta weighting term, which in return is more difficult to fit. In comparison, the weighting of the weight loss behaves more stable and can be approximated more easily.

The quality of the regressed functions in Figure 5.17 are examined in Figure 5.18 by plotting the theoretical achieved compression for a given hyperparameter, as dictated by the predictor functions, against true compression results. For this purpose, the prediction functions are used to determine the corresponding hyperparameters for various target compression ratios. These hyperparameters are then used to train and prune different fv-SRN instances, while keeping all other parameters constant across each run. For each compression-hyperparameter pair, three instances are run to observe the training variance of fv-SRN.

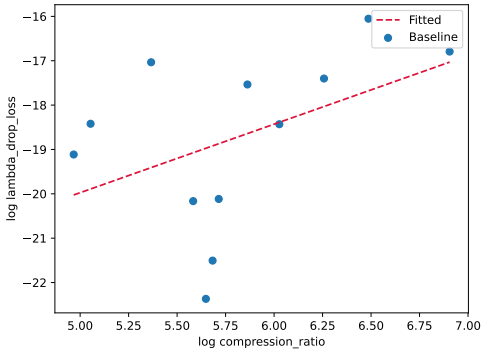
As the final compression rates of each instance corresponding to the same compression-hyperparameter pair are very close to each other, it can be determined that the training process involving fv-SRN and pruning is relatively stable. However, this does not hold for the NAS task of finding optimal parameterizations for the network and pruning algorithms. As seen in Figure 5.14 and Figure 5.17, the data along the pareto frontier exhibits considerable variance, and the high dimensionality of the hyperparameter searchspace creates a complex influence structure of each hyperparameter on the final compression ratio. These factors reduce the ability of the regressed predictor functions to select correct choices of hyperparameters values according to a given compression ratio.

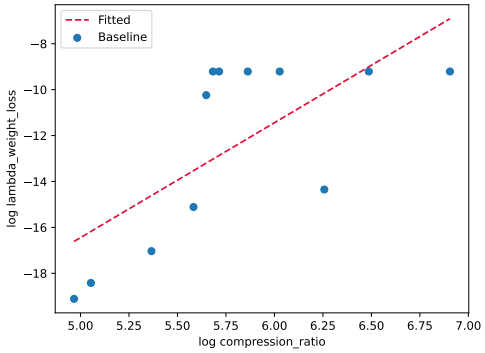
From all pruning methods the regressed function for static variational dropout (see Figure 5.18a) performs the best and the true compression results follow the approximation relatively closely. This observation is consistent with the reasonably stable distribution of compression results in the pareto frontier of static variational dropout, which does not have many outliers and is therefore comparatively easy to



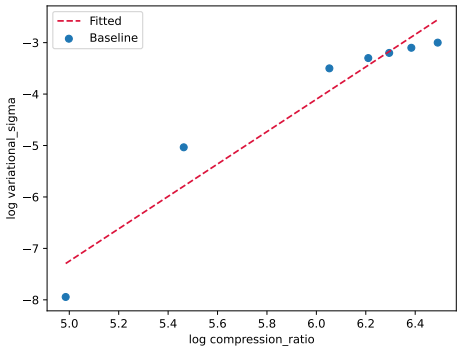


(a) Smallify

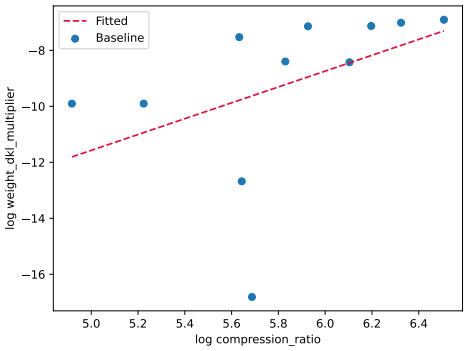




(b) Binary Masking



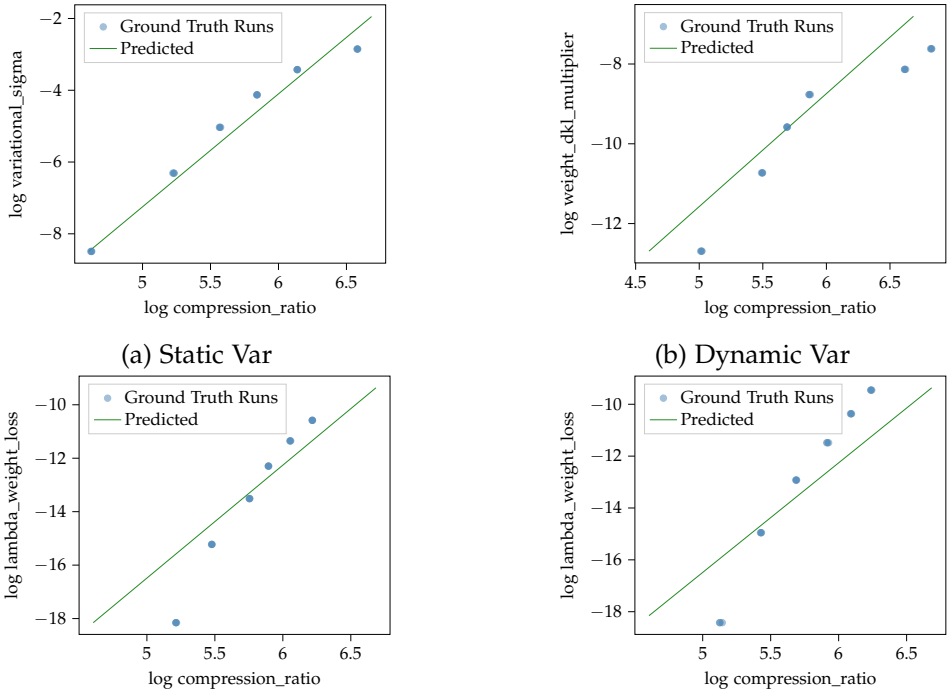
(c) Static Var



(d) Dynamic Var

Figure 5.17.: Regression of simple linear functions that calculate choices of hyperparameters to arrive at a given compression ratio for the Smallify and variational dropout methods. All models analyze the pareto frontier of compression runs on the mhd_p data set with fv-SRN and are fitted in log space.

approximate in a reliable manner. On the other hand the true compression results of the dynamic variational dropout (Figure 5.18b), Smallify (Figure 5.18c) and the simple binary masking (Figure 5.18d) diverge from the prediction functions, although the general trend is still captured.



(a) Static Var

(b) Dynamic Var

(c) Smallify

(d) Binary Masking

Figure 5.18.: Control quality of regressed linear models for fv-SRN. The predicted results of each linear model are compared with real compression outcomes.

As with Neurcomp, the correlations between the final compression rates after pruning and the initial complexity for fv-SRN are also examined. This experiment is depicted in Figure 5.19, by plotting the final compression ratio of a network instance on the x-axis and the initial network compression ratio on the y-axis. The pareto frontier of the hyperparameter search from Figure 5.14 is analyzed and linear regression is used to interlink the final compression and initial compression for each pruning algorithm. Additionally, the influence of feature size and grid size on the final compression is investigated.

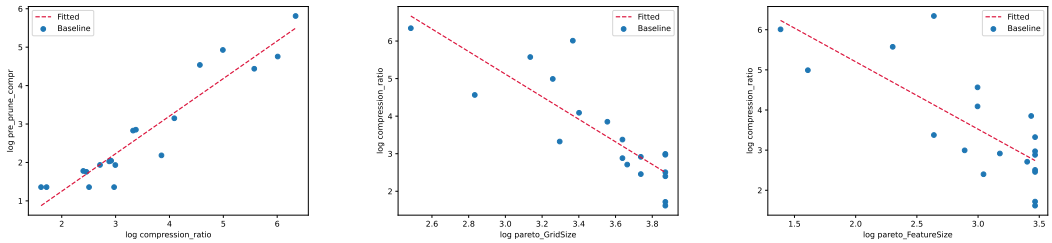
All pruning methods express a strong correlation between their final compression ratio and initial network complexity. Because all pruning algorithms express a function that maps a final compression ratio to a substantially smaller initial compression

ratio, it can be concluded that the fv-SRN undergoes a relatively aggressive pruning process. Compared to the results of the same experiment for Neurcomp (see Figure 5.8), the pruning algorithms on fv-SRN prune larger parts of the network even at smaller compression ratios and do not change their pruning behaviour at different compression ratios (e.g. prune larger chunks at larger compression ratios). Moreover, all pruning methods show a strong correlation of the final compression rate as a function of the feature grid size, while the choice of feature dimension seems rather arbitrary in comparison. These findings are consistent with the results of the original fv-SRN by Weiss et al. [WHW22].

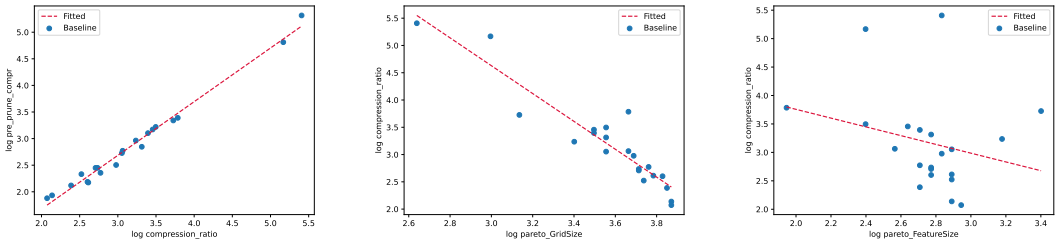
5.4.1. Influence of Wavelet Transform on Pruning

The pruning results depicted in Figure 5.3 and Figure 5.14 demonstrate different affinities to enhancement by network pruning for Neurcomp and fv-SRN. While Neurcomp is not able to achieve significant quality improvements through the use of pruning algorithms, fv-SRN can boost its quality by up to 5 PSNR points. To better understand this behaviour, an analysis is conducted on the performance of the pruning layers in Neurcomp and fv-SRN. Figure 5.20 displays the distribution of the internal per element beta-values for Smallify and dropout-rates for variational dropout on Neurcomp and fv-SRN as layerwise histograms. All distributions are gathered from the corresponding network configurations on the pareto frontiers at a constant final compression ratio of 100, and no tools to further enhance the convergence of the pruning layers, such as an Entropy loss, are used. It can be observed that the pruning layers in fv-SRN consist of a significantly greater number of elements compared to the pruning layers in Neurcomp. Additionally, for fv-SRN the majority of the elements in the layers converge better towards the desired extreme values of each pruning method than in Neurcomp. This effect can be strongly seen for the variational dropout methods. Dynamic and static variational dropout aim to categorize nodes into two groups: Nodes that contain significant information should converge to a droprate of zero and are supposed to be preserved, while nodes that converge to one can be pruned without largely affecting the performance. On the other hand, Smallify is focused on eliminating nodes by reducing the beta values of insignificant nodes to zero, and removing nodes that oscillate near zero. Stronger convergence towards these extreme values enables the pruning algorithms on fv-SRN to better identify unimportant nodes in the network, leading to more informed pruning decisions that remove larger portions of the network while having minimal impact on the overall reconstruction quality.

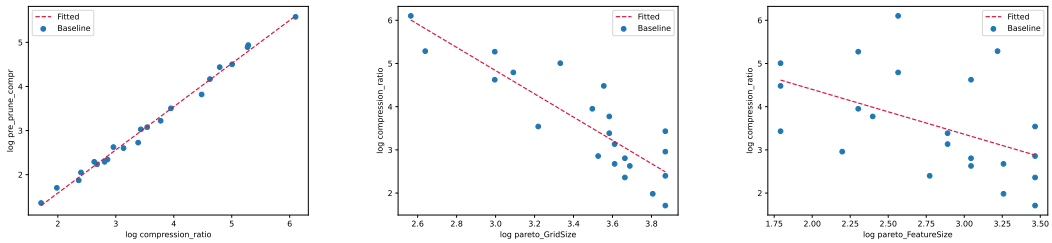
One reason for this behaviour could be that the pruning algorithms in fv-SRN have greater control over most of the network parameters through the latent feature grid, which is entirely observable to the pruning algorithms. In contrast, Neurcomp relies



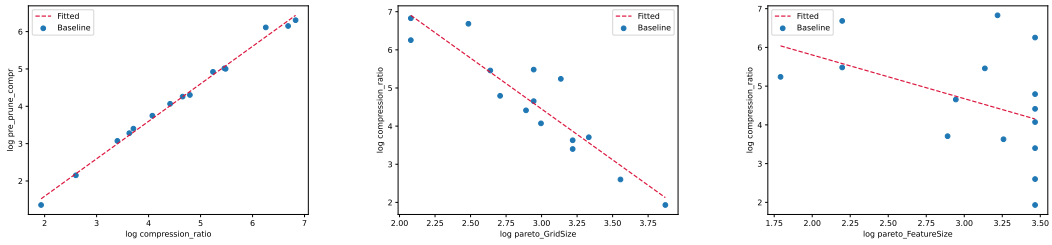
(a) Static Var



(b) Dynamic Var



(c) Smallify



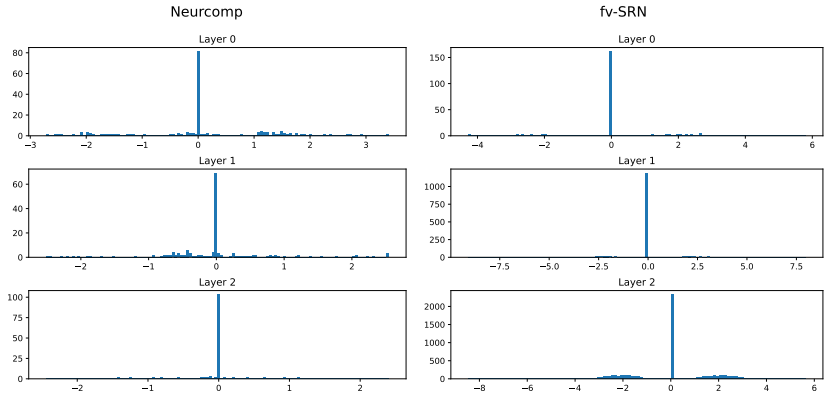
(d) Binary Masking

Figure 5.19.: Correlate the final and initial network compression ratio for variational dropout, Smallify and simple binary masking. Furthermore the influence of the feature size and feature grid size on the final compression are analysed. All data points were taken from the pareto frontier of a free hyperparameter search of fv-SRN on the mhd_p data set.

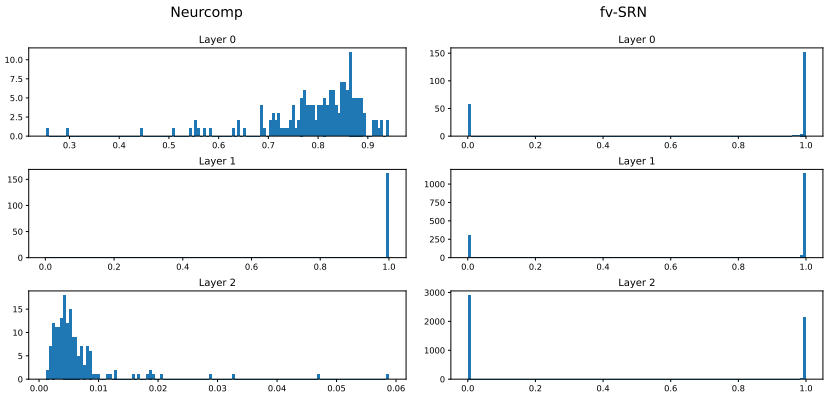
heavily on residual blocks and can only observe a small part of the network domain through the pruning algorithms, limiting its effectiveness in pruning.

Another reason for the observed performance gap in the pruning algorithms for Neurcomp and fv-SRN can be found in the utilization of the wavelet transform on the feature grid of fv-SRN. To this end the influence of the wavelet transform on the effects of pruning on the fv-SRN network is analyzed. This involves comparing the compression performance and pruning behavior of the network using the pruning algorithms, both with and without the frequency transformation applied to the latent feature grid. The results of this experiment are depicted in Figure 5.21. For the experiment NAS is performed to find a pareto frontier for each pruning method of the fv-SRN network in regard to compression rate and PSNR. A fixed set of network architectures along the pareto frontier is selected, and each experiment is conducted either in frequency space with the wavelet transformation or spatially without the wavelet transform. Only the pruning hyperparameters are optimized again with NAS. Then the quality-compression ratio of fv-SRN with and without usage of the wavelet transformation are plotted and compared. In order to analyze the impact of the frequency domain on pruning behavior, the percentage of pruned network parts in relation to the full network size is also measured. A higher amount of pruning percentage means that a higher amount of the original network representation is pruned. The advantages of the wavelet transformation are analyzed on the Smallify (Figure 5.21a), dynamic variational dropout (Figure 5.21b) and static variational dropout (Figure 5.21c) algorithms.

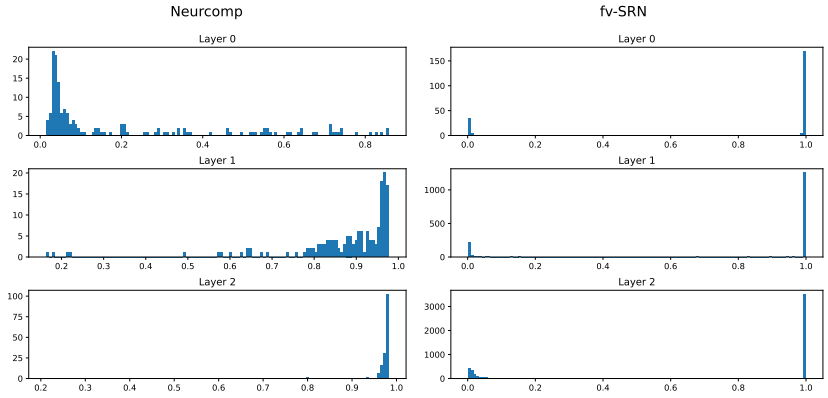
One can see that the quality-compression ratio of the pruned networks with the use of wavelet transformations consistently surpass the networks that do not use the wavelet transformation. In the same way the networks with the use of wavelet transformation prune larger amounts of the original network and can surpass the pruning amount of the networks without wavelet transform by 50%. These results indicate that the frequency representation of the networks feature space is indeed beneficial to pruning and results in more compact network representation than when handling the spatial grid directly.



(a) Smallify

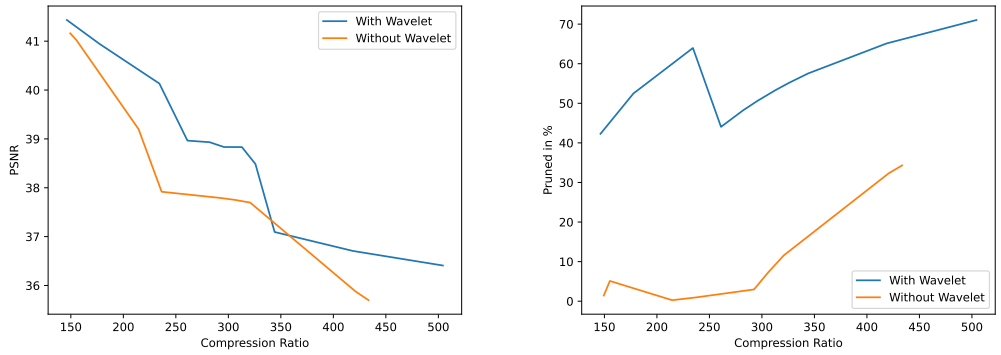


(b) Dynamic Var

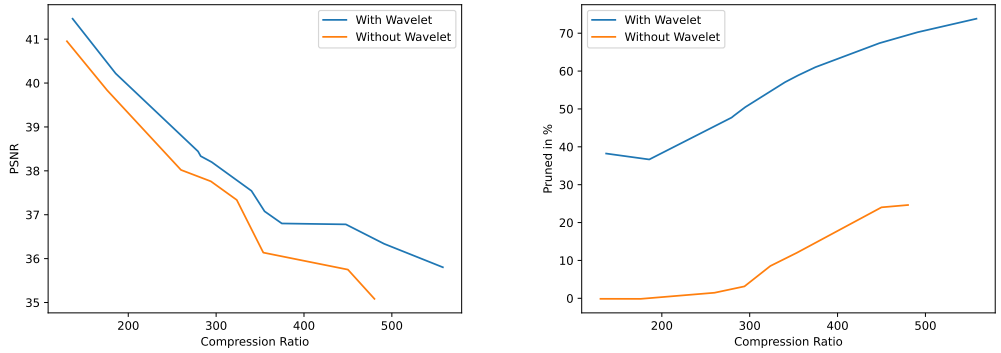


(c) Static Var

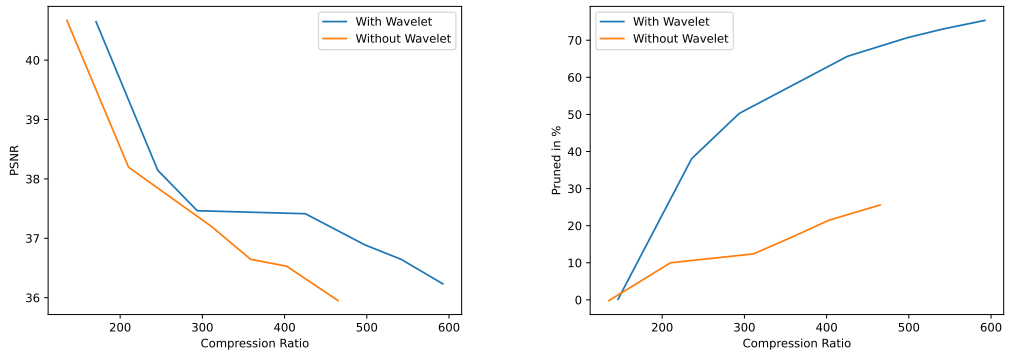
Figure 5.20.: Distribution of pruning values in Neurcomp and fv-SRN. The distributions are gathered from the network pareto frontiers at a constant final compression ratio of 100.



(a) Smallify



(b) Dynamic Var



(c) Static Var

Figure 5.21.: Comparison of pruning effectiveness on the fv-SRN, when the feature grid is represented in the frequency domain or spatial (without wavelets). Each pruning algorithm is compared in regard to its overall quality-compression ratio (left figure) and amount of pruning performed (right figure).

6. Discussion

The main goal of this thesis is to improve implicit neural compression algorithms, such as Neurcomp and fv-SRN, through the utilization of pruning methods like Smallify and variational dropout. These algorithms aim to improve the quality-compression ratio of the underlying network by pruning parameters with small influence to the overall prediction accuracy, thereby generating a sparse and compressed network architecture. Additionally, by using the wavelet transformation to acquire a frequency representation of a networks feature space, this thesis aims to enhance the sparsification capabilities of the pruning algorithms further.

6.1. Effects of Pruning on Neurcomp

As demonstrated in section 5.3 both Smallify and variational dropout exhibit only marginal improvements, if any, in terms of quality-compression ratio over an unpruned network for Neurcomp.

On the turbulence volume data set no pruning method is able to surpass the unpruned baseline. Additionally, the high dimensional search space for hyperparameters complicates finding conclusive pareto frontiers with NAS, leading to considerable noise and variance in the results.

On the larger mhd_p data set the pruning methods exhibit a stronger potential to improve the quality of the unpruned baseline. Smallify and dynamic variational dropout produce marginal improvements, which are not significant in light of the NAS variance. Only static variational dropout is able to produce consistent improvements over the unpruned baseline of 1 to 2 PSNR points. However, given the high variance associated with the NAS, even these quality gains should be approached with caution. Overall these initial results do not demonstrate high enough improvements to justify the usage of pruning algorithms with Neurcomp, considering the additional effort in implementation and more complex hyperparameter optimization.

Hyperparameter analysis is conducted to get more insight into the internal workings of the pruning methods and to establish simple predictor models that can map a desired compression ratio to a selection of a hyperparameter. Indeed, hyperparameter analysis reveals that the behavior of the pruning algorithms, like the magnitude of pruning, is

primarily controlled by one or two hyperparameters. Based on these findings simple predictor functions are regressed. To assess the quality of these functions, the true achieved compression ratios are compared to the predictor's estimates. The results indicate that the prediction models for Smallify and dynamic variational dropout can only provide rough approximations of the compressive behaviour of these pruning algorithms. Nevertheless, static variational dropout is closely matched by its prediction model. The comparatively stable behaviour of static variational dropout could be the reason for its superior performance during pruning, since NAS is able to find the optimal parameterizations more easily.

Finally, additional methods to increase the performance of the pruning algorithms are investigated. Efforts to increase the pruning performance, such as removing the residual blocks from the Neurcomp architecture or incorporating an Entropy loss to improve dropout convergence in variational dropout, do not lead to the desired improvements. While the removal of residual blocks from the Neurcomp architecture enables more pruning options and enhances the overall gain of each pruning algorithm to the unpruned baseline, the loss in prediction accuracy due to the missing residual blocks nullifies this advantage. In the same way the inclusion of an additional Entropy loss is able to increase the convergence of the dropout layers to the extreme values of 0 and 1, but ultimately fails to consistently outperform the dropout version without Entropy loss. One possible explanation for this outcome is that incorporating too many simultaneous losses bloat the network training optimization and complicates the search for optimal hyperparameters, thereby outweighing the advantages of slightly better dropout convergence.

Lastly, based on the observation that different layers of the Neurcomp architecture converge in different ways for variational dropout, the usage of different dropouts per layer instead of a global dropout is proposed. Evaluation of the effects of various per-layer pruning thresholds on the quality-compression ratio of Neurcomp supports this hypothesis. However, implementing layer-wise thresholds for each layer adds additional hyperparameters to the network, which makes it more difficult for a NAS algorithm to identify the best network architecture. A solution to this problem is to choose individual layer pruning thresholds only after initial training of the network has completed. This way, the dimensionality of the hyperparameter space is not bloated and the effectiveness of different layer thresholds can be evaluated more efficiently without the need for retraining the model from scratch.

6.2. Effects of Pruning in fv-SRN

As demonstrated in section 5.4, fv-SRN seems to be more suitable for enhancing compressive abilities of the base network with pruning algorithms.

The pareto frontiers of the pruning algorithms are able to surpass the unpruned baseline by 2-5 PSNR points for small compression ratios on the turbulence volume data set. While the pruning algorithms can also outperform the unpruned baseline at larger compression ratios in places, the high variance observed in the NAS makes conclusive decisions about the advantage of the pruning algorithms at larger compression ratios unfeasible. The variance and noise exhibited by the fv-SRN experiments when tasked to find optimal pareto frontiers with NAS is significantly higher than the variance observed with Neurcomp. Besides differences in network structure and training process, the high variance of hyperparameter search in fv-SRN can likely be attributed to its more complex hyperparameter search space, which includes parameters describing the feature vector and grid sizes, in addition to the fully connected network.

For the larger mhd_p data set, the pruning algorithms are able to realise even higher quality-compression improvements of up to 4 to 5 PSNR points compared to the unpruned baseline. Compared to the smaller datset, the pruning methods are able to consistently outperform the baseline, even at higher compression ratios. Although Smallify performs the best of all dropout algorithms for the mhd_p data set, the differences in quality are relatively minor. Additionally, the variational dropout algorithm does also show promising improvements on the mhd_p data set and is even able to outperform Smallify for some compression ranges in the turbulence volume data set. Because of the high variance present in the data it is hard to arrive at a definitive conclusion regarding the most effective pruning algorithm for fv-SRN.

As for Neurcomp, a hyperparameter analysis is conducted on the pruning pareto frontiers provided by NAS in order to find the most influential hyperparameters for each pruning method. Based on these findings simple linear predictor functions are generated that map a given selection for a final compression ratio to a corresponding choice of pruning hyperparameter. With the help of these simple functions the arduous task of tuning the network- and pruning hyperparameters to fit the desired final compression ratio could be simplified. However, this goal is complicated by the high variance present in the data of the pareto frontiers for fv-SRN. Although static variational dropout is again closely matches the predictive model, all pruning methods are not as accurately captured by the prediction models as observed in Neurcomp.

Despite the high variance present in NAS, fv-SRN displays promising quality gains when coupling the network with pruning methods. These results support the initial

idea of enabling a SRN to learn an optimal network size in regard to both reconstruction quality and compression ratio with network pruning techniques. Although the pruning algorithms, especially variational dropout and Smallify, have positive impacts on the overall quality of data compression with fv-SRN, real world applicability of these methods is negatively affected by the high variance when tasked to find optimal network and pruning parameterizations. This high variability makes it challenging for users to find the best hyperparameters for fv-SRN and the pruning algorithms, which hinders the precise computation of the final compression ratio and reconstruction accuracy post pruning. Future research could further improve the quality-compression ratio of pruning on fv-SRN by limiting the dimensionality of the hyperparameter searchspace for NAS, or by providing a more stable training of the pruning algorithms. For variational dropout this could be achieved through implementation of the local reparameterization trick [KSW15] or better suited priors, as proposed by Nguyen et al. [Ngu+21].

6.3. Comparison of Pruning on Neurcomp and fv-SRN

Neurcomp and fv-SRN seem to have varying degrees of suitability for improving compression quality through pruning algorithms. While the search for the pareto frontiers of the pruning algorithms on Neurcomp behaves more stable and exhibits less variance, only pruning on fv-SRN leads to significant improvements of the quality-compression ratio of the network. Figure 5.20 indicates that the pruning values generated by the pruning layers converge more effectively to the target extreme values of each pruning technique in fv-SRN than in Neurcomp. This superior convergence capacity enables the pruning algorithms in fv-SRN to recognize unimportant nodes in the network more accurately, resulting in better pruning decisions. As seen for Neurcomp in Figure 5.3 and Figure 5.4, as well as for fv-SRN in Figure 5.14 and Figure 5.15, fv-SRN thus demonstrates considerably higher quality-compression gain from the usage of additional pruning algorithms in the network. While the pruning algorithms are able to improve the PSNR at most 1 to 2 points for Neurcomp, fv-SRN is able to obtain PSNR improvements between 4 to 5 points when compared to unpruned baselines of the corresponding networks. Even though generation of the pareto frontiers with NAS exhibits considerably more variance for fv-SRN, the additional quality gain makes it the better alternative for optimization with pruning algorithms.

One reason for this behaviour could be the different pruning approaches for Neurcomp and fv-SRN: Pruning in Neurcomp is targeting the network layers and is thus restricted by the residual blocks. While it is possible to remove the residual blocks, opening the entire network to pruning, doing so results in a considerable loss of stability

for the network and negates any potential performance gains from pruning. On the other hand pruning in fv-SRN is aiming at sparsifying the latent feature grid, where the whole grid domain is more accessible to the pruning algorithms.

The superior pruning performance of fv-SRN can also be attributed to the use of the wavelet transformation for representation of the feature grid. By transforming the feature grid into the frequency domain, different transform coefficients capture distinct low-frequency and high-frequency information about the original data. Notably, a majority of the original signal's energy is concentrated in a small number of coefficients. This trait helps the pruning algorithms to identify the transform coefficients with the most influence to prediction accuracy and enables the algorithms to more easily discards the less important coefficients. The sparsity potential of the wavelet transform thus exceeds that of the spatial representation, making the pruning methods more efficient.

All pruning methods on the Neurcomp and fv-SRN achieve a higher quality gain when tasked to prune on larger data sets and larger networks, as opposed to smaller data sets and smaller networks. This indicates that the size of the encoding network is crucial for the success of the pruning algorithms. If a pruning algorithm is applied to a large neural network, it will be able to find and remove unimportant nodes more easily compared to a small network. Conversely, in a smaller network, a pruned node will have a greater impact on the overall quality of the network, thereby decreasing the effectiveness of the pruning. As a result, pruning algorithms should be preferably paired with SRN when tasked to handle large data, but can lead to no visible quality-compression gain when compressing small data sets.

7. Conclusion

This thesis implements and investigates three pruning methods, namely a simple binary masking, Smallify [Lec+18], and variational dropout [MAV17], to explore their potential in improving the compression quality of Neurcomp [Lu+21] and fv-SRN [WHW22]. While the pruning of the Neurcomp architecture does not result in notable improvements, the pruning algorithms are able to enhance the quality of fv-SRN. The analysis in this thesis concludes that network pruning algorithms are most effective in enhancing the quality of networks where a high number of parameters are susceptible to pruning. This observation is supported by the fact that pruning algorithms achieve better quality improvements when used with networks that handle larger data sets and thus boast a higher parameter count in the network. Essentially, the higher the initial parameter count of the model, the greater the potential for improvements through pruning. The effectiveness of the pruning algorithms can be further optimized by applying frequency transformations to the feature space of the network, thereby enabling a more efficient sparsification compared to the untransformed spatial representation. However, the applicability of the findings in this thesis is reduced by the large variance of compression results observed during NAS. Finding optimal hyperparameters for a fv-SRN instance and pruning algorithm, accurately predicting the pruning behavior and thus the final compression ratio and quality, becomes challenging due to this factor.

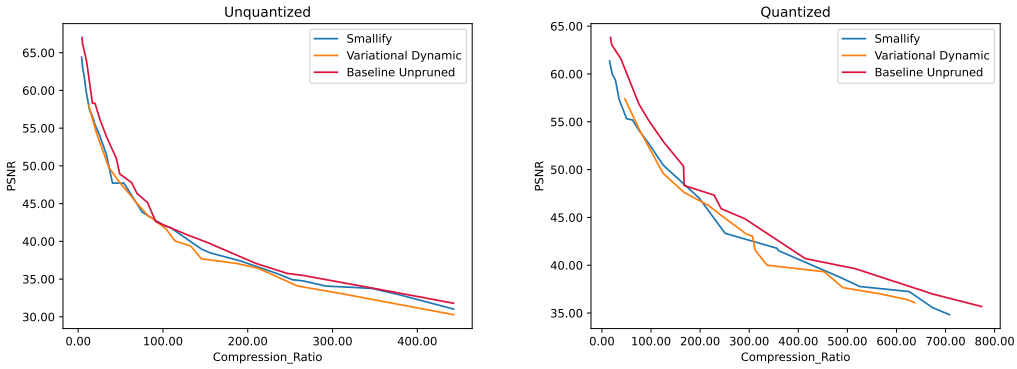
There still exist several open issues and untested methods that could potentially improve the stability and compression quality of pruning methods on SRN. In particular, variational dropout has been the focus of recent research on pruning algorithms, and studies have been conducted that are able to improve the performance of the dropout algorithm. The local reparameterization trick proposed by Kingma et al [KSW15], for example, represents an efficient tool for training of the variational dropout layers, and improves the estimation of the gradients for the expected log likelyhood during training. Instead of sampling the separate variational noise per input and applying the noise to the output of a previous layer to get the activation values of the dropout layer, the outputs can be sampled directly from their implied Gaussian distributions. This procedure yields a gradient estimator that is computationally efficient and has lower variance, thereby improving training consistency.

Other work includes Variational Structured Dropout (VSD) by Nguyen et al. [Ngu+21]. Their proposal involves using Householder transformations on the multiplicative

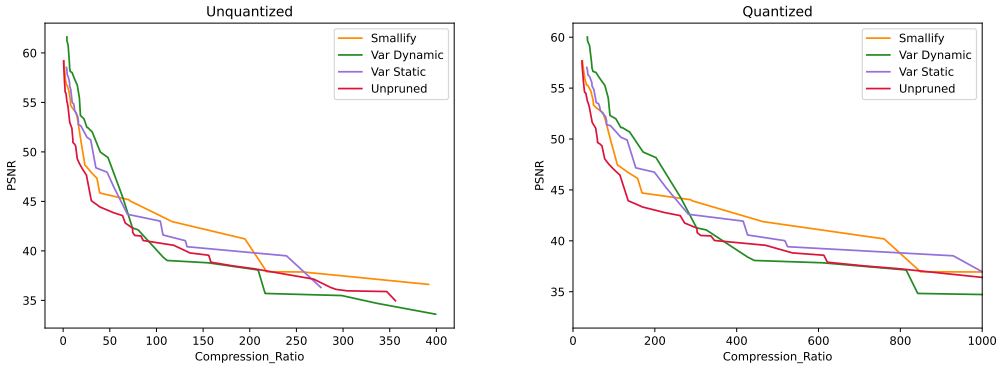
noise for learning a structured representation of the variational Gaussian noise. This method obtains an approximate posterior with structured covariance, which enables the posterior to capture dependencies among network weights and is thereby able to avoid limitations associated with the mean-field Gaussian priors used in the implementation of this thesis.

A. Figures

A.1. Pruning With and Without Quantization



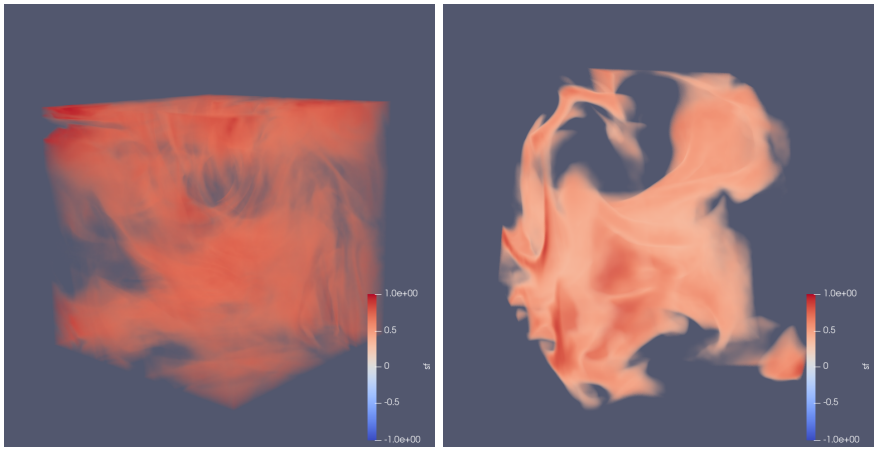
(a) Neurcomp



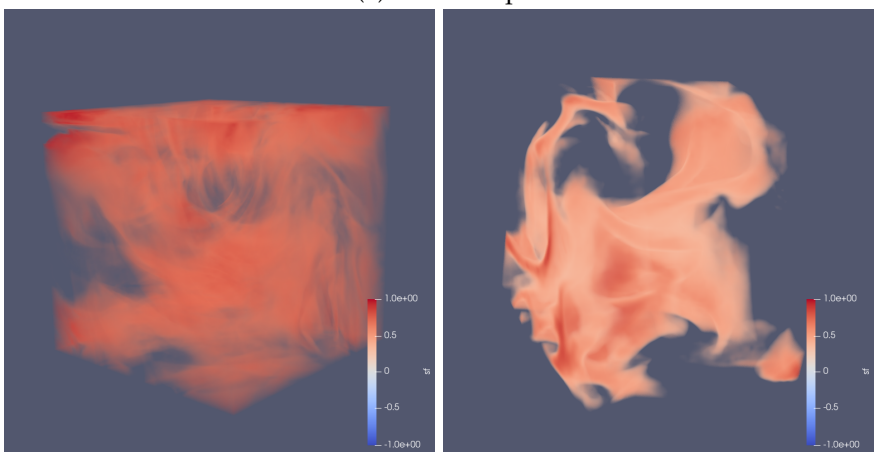
(b) fv-SRN

Figure A.1.: Evaluating the quality-compression ratio of the pruning algorithms on Neurcomp (a) and fv-SRN (b) prior and post quantization on the test data set. Even though quantization affects the final compression ratio, the relations among the plots do not change.

A.2. Rendering of Data Sets



(a) Neurcomp



(b) fv-SRN

Figure A.2.: Rendering of the mhd_p data set (255, 255, 255) on the left and turbulence data set (150, 150, 150) on the right. Both data sets are compressed by Neurcomp (a) and fv-SRN (b) at a compression ratio of 100. Neurcomp and fv-SRN compress the mhd_p data set at 44 PSNR, while the turbulence data set is compressed at 42.2 PSNR by Neurcomp and at 41.9 PSNR by fv-SRN.

List of Figures

List of Tables

Glossary

activation function The activation function of a node defines the output of that node on a series of inputs. Often the activation function is nonlinear, in order to allow the networks to compute nontrivial problems that don't have to follow linear behaviour. 10, 12, 20, 35

arithmetic coding Arithmetic coding is an entropy based compression algorithm that encodes frequently occurring symbols with fewer bits than not so frequent symbols. Different to other types of entropy coders, arithmetic coding encodes the input data as a single number of arbitrary precision, from which the original data can be restored. 6

backpropagation Algorithm for training feedforward networks. This is done by computing the gradient of a given loss function with respect to the network weights for an observed input-output pair. The algorithm iterates backwards through the network and calculates the gradient of each layer, given the gradient of the previous layer. According to a calibrated learning strategy, the weights of each layer are then updated, to minimize their impact on the loss.. 13, 16, 18

basis wavelet Mother wavelet that is translated and dilated to form smaller wavelets that extract specific information from a base signal. 8

bayesian optimization Global optimization strategy that employs a surrogate model and an acquisition function to probe a black-box function in a sample-efficient manner. 25, 26

bias Bias are parameters of a neural network that are added to the weighted input of a node to shift the activation function. 10, 12

bit-plane coding Compression technique that uses the binary representation of the input data and iteratively compresses the same bit (bit-plane), starting from the most significant bit-plane. 6

chain rule Formula that is used to describe the derivative of the composite of two functions $f(\cdot)$ and $g(\cdot)$ as $f(g(x))' = f'(g(x)) \cdot g'(x)$.. 18

direct volume rendering A widely used visualization method for rendering volume data. The technique works by using raycasting to scan the volume along rays $r(t)$ from a viewing diretion $v = (v_1, v_2)$. Sampled points along the ray are evaluated by means of interpolation, associated color and opacity values $c(r(t), v)$ and $\sigma(r(t))$ are read out, shaded and occluded according to the accumulated density along the ray $T(t)$ and then calculated to the resulting pixel value $C(r)$:

$$C(r) = \int_{t1}^{t2} T(t) \cdot \sigma(r(t)) \cdot c(r(t), v) \tag{A.1}$$

. 4, 11, 14, 16, 29, 30

expected log-likelihood The natural logarithm of the likelihood. The likelihood $L_D(\Theta)$ describes the probability of observing data D when parameterizing a probability distribution with Θ. 21

fine-tuning Retraining a network with a lower learning rate to further enhance accuracy of the neural network. 17

HOSVD Many transform-based compression algorithms for volumetric data rely on some form of data-dependent bases that decompose the data set into smaller approximations of lower dimension. HOSVD (Higher-Order SVD) is one such approximation and is a generalization of matrix SVD for tensors. 6

Huffman encoding Huffman encoding is a compression algorithm that uses the entropy of the input data to encode frequently occurring symbols with fewer bits than not so frequent symbols. 34

hyperparameter Hyperparameters are fixed before training and determine the network architecture (e.g., by specifying the number or size of hidden layers) and the training process (e.g., by deciding how much data to use in each training iteration or how fast to update the network's parameters). 12, 16, 20, 24, 36, 37, 39, 40, 43, 44, 46, 48–56, 58, 61, 63, 65–70, 72, 73, 76–79, 81, 87

k-means clustering Quantization method, where the elements of an input vector are partitioned into k clusters. Each element is then assigned to the cluster with the nearest mean or cluster centroid. Each original element is then represented by the cluster cencer value, thus quantizing the data.. 13, 34

Kullback-Leibler divergence The Kullback-Leibler divergence $D_{KL}(Q||P)$ measures the distance between two probability distributions Q and P. 21

node Also called neuron. A neural network computational unit comprising one or more weighted input connections, an activation function combining the inputs in some manner, and an output connection. 10

pareto frontier Term used in mutli-objective optimization to describe the set of optimal tradeoff solutions (pareto-optimal solutions).. 24, 49, 52–54, 56, 57, 60, 64–66, 68, 70, 78, 79

posterior distribution Conditional probability that results from updating the prior probability $p(\Theta)$ with information from data D: $p(\Theta|D) = p(D|\Theta) \cdot p(\Theta)/p(D)$. Intuitively, this can be understood as asking "What is the probability of Θ, given D". 21

PSNR Logarithmic measure commonly used to quantify the reconstruction quality of data subject to lossy compression. It describes the relationship between the maximum possible power of a signal and the amount of interfering noise that affects the quality of the signal. A high PSNR corresponds to a high quality of the reconstructed signal.. 48–53, 59, 61–63, 65, 66, 73

quantization error Difference between the original value and the mapped value after quantization. 6

quantizer Device that performs quantization and maps a set of input values to a finite set of output values that approximate the input data. 6

run-length coding Run length encoding is a compression method in which sequences containing redundant data are stored as a single data value indicating the repeated block and the number of times it is repeated in the input data. 6

structured pruning Culling of individual parameters of the network weight matrices, resulting in sparse weight matrices. 19, 35, 38

sub-band Frequency bands. 8, 9

targeted pruning Cull neurons adaptively according to a metric that induces a parameter ranking. 19, 34, 38

unstructured pruning Culling of parameter groups of the network weight matrices (e.g., culling entire neurons), resulting in dense weight matrices. 35, 38

variational dropout Extends dropout with a Bayesian approach to parameterize a network using an approximated posterior distribution. This way, the drop rate is not preset as a hyperparameter, but can be learned individually for each layer through training. 20, 37

wavelet Group of oscilating functions that can be fit to match a part of a base input signal. 7

wavelet transform Transform that captures a base signal as a set of wavelets derived from dilating and translating a basis wavelet. 7–9

weight Weights are parameters of a neural network that are multiplied with the input of a node to symbolize the weighted connections between nodes of different layers. 10–13, 20, 24, 35

Acronyms

DNN Deep Neural network. 10, 16, 34, 35, 38

DWT Discrete Wavelet Transform. 8

EMA Exponential Moving Average. 20

NAS Neural Architecture Search. 24, 25, 48–53, 59–61, 63, 65, 66, 68, 73, 76–79, 81, 85

SRN Scene Representation Network. iv, 1, 2, 5, 6, 10, 11, 27, 31, 79–81, 87

Bibliography

[Ahr+05] J. Ahrens, B. Geveci, C. Law, C. Hansen, and C. Johnson. "36-paraview: An end-user tool for large-data visualization." In: *The visualization handbook* 717 (2005), pp. 50038–1.

[Bak+16] B. Baker, O. Gupta, N. Naik, and R. Raskar. "Designing neural network architectures using reinforcement learning." In: *arXiv preprint arXiv:1611.02167* (2016).

[Bak+18] E. Bakshy, L. Dworkin, B. Karrer, K. Kashin, B. Letham, A. Murthy, and S. Singh. "AE: A domain-agnostic platform for adaptive experimentation." In: *Conference on Neural Information Processing Systems*. 2018, pp. 1–8.

[BB86] R. N. Bracewell and R. N. Bracewell. *The Fourier transform and its applications*. Vol. 31999. McGraw-Hill New York, 1986.

[Bha18] J. Bhattacharjee. *fastText Quick Start Guide: Get started with Facebook's library for text representation and classification*. Packt Publishing Ltd, 2018.

[Bla+20] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. "What is the state of neural network pruning?" In: *Proceedings of machine learning and systems* 2 (2020), pp. 129–146.

[BLC13] Y. Bengio, N. Léonard, and A. Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." In: *arXiv preprint arXiv:1308.3432* (2013).

[BLP19] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. "TTHRESH: Tensor compression for multidimensional visual data." In: *IEEE transactions on visualization and computer graphics* 26.9 (2019), pp. 2891–2903.

[BP16] R. Ballester-Ripoll and R. Pajarola. "Lossy volume compression using Tucker truncation and thresholding." In: *The Visual Computer* 32.11 (2016), pp. 1433–1446.

[Che+15] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. "Compressing neural networks with the hashing trick." In: *International conference on machine learning*. PMLR. 2015, pp. 2285–2294.

[Che+17] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. "A survey of model compression and acceleration for deep neural networks." In: *arXiv preprint arXiv:1710.09282* (2017).

[Che+21] A. Chen, Z. Xu, F. Zhao, X. Zhang, F. Xiang, J. Yu, and H. Su. "Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 14124–14133.

[CK19] R. M. Cichy and D. Kaiser. "Deep neural networks as scientific models." In: *Trends in cognitive sciences* 23.4 (2019), pp. 305–317.

[CZ19] Z. Chen and H. Zhang. "Learning implicit fields for generative shape modeling." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5939–5948.

[DBB21] S. Daulton, M. Balandat, and E. Bakshy. "Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 2187–2200.

[DC16] S. Di and F. Cappello. "Fast error-bounded lossy HPC data compression with SZ." In: *2016 ieee international parallel and distributed processing symposium (ipdps)*. IEEE. 2016, pp. 730–739.

[dCM00] O. de Vel, D. Coomans, and Y. Mallett. "Chapter 19 - Wavelet-Based Image Compression." In: *Wavelets in Chemistry*. Ed. by B. Walczak. Vol. 22. Data Handling in Science and Technology. Elsevier, 2000, pp. 457–478. DOI: https://doi.org/10.1016/S0922-3487(00)80044-1. URL: https://www.sciencedirect.com/science/article/pii/S0922348700800441.

[Den+22] K. Deng, A. Liu, J.-Y. Zhu, and D. Ramanan. "Depth-supervised nerf: Fewer views and faster training for free." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12882–12891.

[Eri+21] D. Eriksson, P. I.-J. Chuang, S. Daulton, P. Xia, A. Shrivastava, A. Babu, S. Zhao, A. Aly, G. Venkatesh, and M. Balandat. "Latency-aware neural architecture search with multi-objective bayesian optimization." In: *arXiv preprint arXiv:2106.11890* (2021).

[Fin09] T. Finch. "Incremental calculation of weighted mean and variance." In: *University of Cambridge* 4.11-5 (2009), pp. 41–42.

[Gal+16] Y. Gal et al. "Uncertainty in deep learning." In: (2016).

[Gao+22] K. Gao, Y. Gao, H. He, D. Lu, L. Xu, and J. Li. "Nerf: Neural radiance field in 3d vision, a comprehensive review." In: *arXiv preprint arXiv:2210.00379* (2022).

[Gar+21] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin. "Fastnerf: High-fidelity neural rendering at 200fps." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 14346–14355.

[Gom+19] A. N. Gomez, I. Zhang, S. R. Kamalakara, D. Madaan, K. Swersky, Y. Gal, and G. E. Hinton. "Learning sparse networks using targeted dropout." In: *arXiv preprint arXiv:1905.13678* (2019).

[Gon+14] Y. Gong, L. Liu, M. Yang, and L. Bourdev. "Compressing deep convolutional networks using vector quantization." In: *arXiv preprint arXiv:1412.6115* (2014).

[Han+20] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu. "Ghostnet: More features from cheap operations." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 1580–1589.

[He+16] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[Hin+12] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors." In: *arXiv preprint arXiv:1207.0580* (2012).

[HJ11] C. D. Hansen and C. R. Johnson. *Visualization Handbook*. Academic Press, 2011.

[HMD15] S. Han, H. Mao, and W. J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." In: *arXiv preprint arXiv:1510.00149* (2015).

[HMG18] J. Hron, A. Matthews, and Z. Ghahramani. "Variational Bayesian dropout: pitfalls and fixes." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2019–2028.

[HVD15] G. Hinton, O. Vinyals, and J. Dean. "Distilling the knowledge in a neural network." In: *arXiv preprint arXiv:1503.02531* (2015).

[HWW22] K. Höhlein, S. Weiss, and R. Westermann. "Evaluation of Volume Representation Networks for Meteorological Ensemble Compression." In: (2022).

[Jos+22] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun. "Hands-on Bayesian neural networks—A tutorial for deep learning users." In: *IEEE Computational Intelligence Magazine* 17.2 (2022), pp. 29–48.

[KA01] T. Kim and T. Adali. "Approximation by fully complex MLP using elementary transcendental activation functions." In: *Neural Networks for Signal Processing XI: Proceedings of the 2001 IEEE Signal Processing Society Workshop (IEEE Cat. No. 01TH8584)*. IEEE. 2001, pp. 203–212.

[Kan+18] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. "Neural architecture search with bayesian optimisation and optimal transport." In: *Advances in neural information processing systems* 31 (2018).

[KB14] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[KSW15] D. P. Kingma, T. Salimans, and M. Welling. "Variational dropout and the local reparameterization trick." In: *Advances in neural information processing systems* 28 (2015).

[Lak+13] S. Lakshminarasimhan, N. Shah, S. Ethier, S.-H. Ku, C.-S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. "ISABELA for effective in situ compression of scientific data." In: *Concurrency and Computation: Practice and Experience* 25.4 (2013), pp. 524–540.

[LC87] W. E. Lorensen and H. E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm." In: *ACM siggraph computer graphics* 21.4 (1987), pp. 163–169.

[Le+22] H. Le, R. K. Høier, C.-T. Lin, and C. Zach. "AdaSTE: An Adaptive Straight-Through Estimator to Train Binary Neural Networks." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 460–469.

[Lec+18] G. Leclerc, M. Vartak, R. C. Fernandez, T. Kraska, and S. Madden. "Smallify: Learning network size while training." In: *arXiv preprint arXiv:1806.03723* (2018).

[Li+08] Li, Perlman, Wan, Yang, Meneveau, Burns, Chen, Szalay, and Eyink. "A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence." In: *Journal of Turbulence* 9.31 (2008), pp. 524–540.

[Li+18] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs. "Data reduction techniques for simulation, visualization and data analysis." In: *Computer graphics forum*. Vol. 37. 6. Wiley Online Library. 2018, pp. 422–447.

[Lin14] P. Lindstrom. "Fixed-rate compressed floating-point arrays." In: *IEEE transactions on visualization and computer graphics* 20.12 (2014), pp. 2674–2683.

[LL16] V. Lebedev and V. Lempitsky. "Fast convnets using group-wise brain damage." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2554–2564.

[LSV19] A. Labach, H. Salehinejad, and S. Valaee. "Survey of dropout methods for deep neural networks." In: *arXiv preprint arXiv:1904.13310* (2019).

[Lu+21] Y. Lu, K. Jiang, J. A. Levine, and M. Berger. "Compressive neural representations of volumetric scalar fields." In: *Computer Graphics Forum*. Vol. 40. 3. Wiley Online Library. 2021, pp. 135–146.

[LW16] C. Louizos and M. Welling. "Structured and efficient variational deep learning with matrix gaussian posteriors." In: *International conference on machine learning*. PMLR. 2016, pp. 1708–1716.

[Mar+21] J. N. Martel, D. B. Lindell, C. Z. Lin, E. R. Chan, M. Monteiro, and G. Wetzstein. "Acorn: Adaptive coordinate networks for neural scene representation." In: *arXiv preprint arXiv:2105.02788* (2021).

[MAV17] D. Molchanov, A. Ashukha, and D. Vetrov. "Variational dropout sparsifies deep neural networks." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2498–2507.

[Mes+19] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. "Occupancy networks: Learning 3d reconstruction in function space." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4460–4470.

[Mil+20] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. "Nerf: Representing scenes as neural radiance fields for view synthesis." In: *European conference on computer vision*. Springer. 2020, pp. 405–421.

[Mül+22] T. Müller, A. Evans, C. Schied, and A. Keller. "Instant neural graphics primitives with a multiresolution hash encoding." In: *arXiv preprint arXiv:2201.05989* (2022).

[Nec04] M. Nechyba. "Introduction to the discrete wavelet transform (DWT)." In: *University of Florida, February* (2004).

[Nek+17] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov. "Structured bayesian pruning via log-normal multiplicative noise." In: *Advances in Neural Information Processing Systems* 30 (2017).

[Ngu+21] S. Nguyen, D. Nguyen, K. Nguyen, K. Than, H. Bui, and N. Ho. "Structured dropout variational inference for Bayesian neural networks." In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 15188–15202.

[Par+19] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. "Deepsdf: Learning continuous signed distance functions for shape representation." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 165–174.

[Rah+19] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. "On the spectral bias of neural networks." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 5301–5310.

[Rea+17] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. "Large-scale evolution of image classifiers." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2902–2911.

[Ren+21] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang. "A comprehensive survey of neural architecture search: Challenges and solutions." In: *ACM Computing Surveys (CSUR)* 54.4 (2021), pp. 1–34.

[Rho+22] D. Rho, B. Lee, S. Nam, J. C. Lee, J. H. Ko, and E. Park. "Masked Wavelet Representation for Compact Neural Radiance Fields." In: *arXiv preprint arXiv:2212.09069* (2022).

[Rom+14] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. "Fitnets: Hints for thin deep nets." In: *arXiv preprint arXiv:1412.6550* (2014).

[Ron+19] B. Ronen, D. Jacobs, Y. Kasten, and S. Kritchman. "The convergence rate of neural networks for learned functions of different frequencies." In: *Advances in Neural Information Processing Systems* 32 (2019).

[Sai+13] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. "Low-rank matrix factorization for deep neural network training with high-dimensional output targets." In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6655–6659.

[SF+00] M. Sonka, J. M. Fitzpatrick, et al. "Handbook of medical imaging. Volume 2, Medical image processing and analysis." In: SPIE. 2000.

[SF03] R. S. Stanković and B. J. Falkowski. "The Haar wavelet transform: its status and achievements." In: *Computers & Electrical Engineering* 29.1 (2003), pp. 25–44.

[Sit+20] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. "Implicit neural representations with periodic activation functions." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 7462–7473.

[Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[Sun+18] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore. "Testing deep neural networks." In: *arXiv preprint arXiv:1803.04792* (2018).

[SV19] H. Salehinejad and S. Valaee. "Ising-dropout: A regularization method for training and compression of deep neural networks." In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 3602–3606.

[SZW19] V. Sitzmann, M. Zollhöfer, and G. Wetzstein. "Scene representation networks: Continuous 3d-structure-aware neural scene representations." In: *Advances in Neural Information Processing Systems* 32 (2019).

[Tak+21] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler. "Neural geometric level of detail: Real-time rendering with implicit 3D shapes." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 11358–11367.

[TM02] D. S. Taubman and M. W. Marcellin. "JPEG2000: Standard for interactive imaging." In: *Proceedings of the IEEE* 90.8 (2002), pp. 1336–1357.

[TST20] M. Tomczak, S. Swaroop, and R. Turner. "Efficient low rank gaussian variational inference for neural networks." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 4610–4622.

[VBU07] C. Vonesch, T. Blu, and M. Unser. "Generalized Daubechies wavelet families." In: *IEEE Transactions on Signal Processing* 55.9 (2007), pp. 4415–4429.

[WHW22] S. Weiss, P. Hermüller, and R. Westermann. "Fast neural representations for direct volume rendering." In: *Computer Graphics Forum*. Vol. 41. 6. Wiley Online Library. 2022, pp. 196–211.

[WM13] S. Wang and C. Manning. "Fast dropout training." In: *international conference on machine learning*. PMLR. 2013, pp. 118–126.

[Wur+21] S. W. Wurster, H.-W. Shen, H. Guo, T. Peterka, M. Raj, and J. Xu. "Deep hierarchical super-resolution for scientific data reduction and visualization." In: *arXiv preprint arXiv:2107.00462* (2021).

[Xie+22] Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, N. Khan, F. Tombari, J. Tompkin, V. Sitzmann, and S. Sridhar. "Neural fields in visual computing and beyond." In: *Computer Graphics Forum*. Vol. 41. 2. Wiley Online Library. 2022, pp. 641–676.

[Yan+22] R. Yang, T. Xiao, Y. Cheng, J. Suo, and Q. Dai. "TINC: Tree-structured Implicit Neural Compression." In: *arXiv preprint arXiv:2211.06689* (2022).

[Yu+21] A. Yu, V. Ye, M. Tancik, and A. Kanazawa. "pixelnerf: Neural radiance fields from one or few images." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 4578–4587.

[ZB21] F. Zhang and D. R. Bull. *Intelligent image and video compression: communicating pictures*. Academic Press, 2021.

[Zho+17] Z. Zhou, Y. Hou, Q. Wang, G. Chen, J. Lu, Y. Tao, and H. Lin. "Volume upscaling with convolutional neural networks." In: *Proceedings of the Computer Graphics International Conference*. 2017, pp. 1–6.